Санкт-Петербургский государственный университет

КУЗНЕЦОВА Дарья Сергеевна

Выпускная квалификационная работа

Решение задач оптимизации роевыми и эволюционными алгоритмами

Уровень образования: магистратура

Направление 01.04.02 «Прикладная математика и информатика»

Основная образовательная программа ВМ.5505. «Математическое и информационное обеспечение экономической деятельности»

Научный руководитель:
доктор физ.-мат наук, профессор
Крылатов Александр Юрьевич

Рецензент:
начальник планово-экономического отдела
ООО «Газпром трансгаз Санкт-Петербург»
Галкин Михаил Эдвардович

Санкт-Петербург
2022

Содержание

Введение

В современной науке наблюдается увеличение количества исследований, посвященных методам роевой оптимизации, одним из которых является алгоритм роя частиц (PSO). Это объясняется развитием окружающих человека систем, где для принятия оптимальных управленческих решений использование классических методов становится невозможным. У целевой функции могут отсутствовать свойства выпуклости, дифференцируемости, унимодальности. Также многие задачи являются NP-трудными и нахождение точного решения на больших размерностях становится невозможным.

Актуальность работы обусловлена усложнением задач оптимизации и как следствие невозможностью их решения точными методами.

Целью работы является сравнительный анализ эффективности роевых и эволюционных алгоритмов в решении прикладных задач двухуровневой оптимизации.

Для осуществления поставленной цели необходимо реализовать следующие задачи:

1. Провести обзор существующих методов оптимизации с применением роевого интеллекта и эволюционных подходов.

2. Осуществить апробацию инструментов роевого интеллекта для решения двухуровневых задач оптимизации на тестовых примерах.

3. Сравнить полученные результаты с эволюционными процедурами решения двухуровневых задач оптимизации на тестовых примерах.

4. Адаптировать наиболее эффективный алгоритм к решению задач двухуровневой оптимизации на транспортных сетях.

5. Реализовать эффективные процедуры решения задачи оптимизации топологии транспортной сети с применением построенных эвристик.

Теоретическая значимость работы заключается в консолидации и систематизации сведений об алгоритме роя частиц, которые в малом объеме представлены в русскоязычной научной литературе. Исследовании новых подходов его адаптации, которые с одной стороны повышают вероятность и

скорость его сходимости, а с другой расширяют круг к различным оптимизационным задачам.

Практическая значимость работы заключается в применении данного алгоритма для получения приближенного решения двухуровневой оптимизационной задачи равновесного распределения транспортных потоков.

Методами исследования настоящей работы являются описание, формализация, классификация, практическое моделирование, эксперимент, сравнение, измерение, итерационные методы, численные методы, обобщение и анализ результатов.

Глава 1. Обзор роевых и эволюционных алгоритмов, описание их модификаций и способов адаптации к задачам оптимизации

1.1. Особенности класса стохастических алгоритмов поиска

Многие проблемы, которые возникают в различных областях знаний, от физики и молекулярной биологии до прикладной математики, можно свести к задачам глобальной оптимизации. Зачастую они имеют высокую вычислительную сложность: у их целевых функций отсутствуют свойства выпуклости, дифференцируемости, унимодальности.

В основе детерминированных эвристических или жадных методов оптимизации лежит принцип нахождения локально лучшего решения на каждом шаге. Поиск глобально лучшего решения обеспечивается через выбор решения, которое является лучшим на всех итерациях [1]. Движение по пространству поиска осуществляется на основании ранее сделанных выборов, а действия на последующих шагах не влияют на текущие, что отличает данных класс алгоритмов от динамического программирования. Как следствие эти методы малозатратны по времени вычисления решения, но не дают возможности гарантировать полученное решение на предмет лучшего из возможных.

Для повышения эффективности нахождения оптимальных решений в 1980-х гг., которая в том числе заключается в сокращении времени работы алгоритма, началась разработка класса стохастических алгоритмов поиска. В разных публикациях они имеют название поведенческих, метаэвристических, инспирированных природой, роевых, популяционных. Особенностью этих алгоритмов является случайность процесса поиска [2].

Стохастические или рандомизированные эвристические методы не дают гарантии получения точного решения, однако при их использовании можно получить близкое к лучшему решение за приемлемое время. Из-за того, что в основе многих подходов лежат наблюдаемые в природе явления, их алгоритмическая реализация является нетривиальным процессом. Для каждой

конкретной задачи оптимизации необходимо исследовать сходимость, быстродействие, эффективность, влияние ее условий и параметров, которые сильно зависят от выбранных эвристик.

Рассматриваемые в настоящей работе алгоритмы роевого интеллекта и эволюционные алгоритмы являются алгоритмами дискретной оптимизации, так как осуществляют поиск лучшего решения итеративно. При этом область использования этих методов включает в себя как непрерывные задачи, так и дискретные и гибридные [3]. В основе этих алгоритмов лежат принципы и закономерности, наблюдающиеся в природе. Они относятся к популяционным методам, так как используют системы, состоящие из агентов.

В основу популяционных алгоритмов легла идея об одновременной обработке нескольких вариантов решения оптимизационных задач, что в свою очередь стало альтернативной траекторным поисковым алгоритмам, в которых происходит эволюционирование только одного кандидата на решение. Все популяционные алгоритмы относятся к классу эвристических, их сходимость не доказана, но в большинстве случаев они дают хорошее решение.

Под агентом чаще всего понимается точка на пространстве решений и поиск лучшего осуществляется через перемещение агентов по этому пространству. В учебнике Карпенко А. П. описаны основные этапы процесса оптимизации всеми популяционными алгоритмами [2]. Его общая схема представлена на Рисунок *1*. На первом этапе происходит инициализация популяции. Она заключается в создании на пространстве поиска заданного числа приближений к искомому решению. На втором этапе осуществляется миграция агентов популяции путем их перемещения с помощью набора специфических миграционных операторов по пространству решений для приближения к экстремуму оптимизируемой функции. На третьем этапе проверяются условия прекращения работы алгоритма. В случае их выполнения лучшее положение агентов принимается за приближенное решение задачи, иначе процесс поиска продолжается и алгоритм возвращается к выполнению второго этапа.

Рисунок 1 – Общая схема поиска приближенного решения оптимизационных задач популяционными алгоритмами

Подход эволюционных методов оптимизации заключается в создании новой популяции агентов на каждой итерации, используя накопленный предыдущими популяциями опыт. В свою очередь, методы роевого интеллекта используют косвенный обмен информацией между участниками роя и перемещают агентов одной популяции по пространству поиска, не создавая новых. В статье Матренина П. В. и Секаева В. Г. [4] предложен способ отнесения алгоритмов к роевым: в формулах, которые описывают миграцию агентов роя, необходимо наличие объекта, который дает возможность косвенного обмена информацией между ними.

Области применения стохастических методов оптимизации имеют широкий спектр. Укрупненно можно выделить такие задачи как:

1. Комбинаторные задачи производственного планирования и логистики, в том числе решение задачи коммивояжера, задачи о назначениях, календарного планирования, транспортной задачи и т. д..

2. Задачи построения телекоммуникационных сетей, что включает в себя оптимизацию управления потоками данных в сети, разработку структуры сети,

определении наиболее эффективных схем вида сетей.

3. Задачи управления автоматизированными системами, например, контроль работы двигателей, разработка контроллеров.

4. Задачи по анализу изображений и видео, в том числе распознавание лиц, сегментация и слияние изображений, распознавание символов.

5. Задачи диагностики болезней и анализа генетических последовательностей.

Также к областям применения стохастических алгоритмов относится энергетика, компьютерная графика, робототехника, обработка сигналов, системы искусственного интеллекта и машинное обучение.

Таким образом, была обозначена проблема усложнения задач оптимизации и как следствие развитие методов их решения, одним из которых является класс стохастических алгоритмов поиска. Была представлена основная идея популяционных алгоритмов, как представителей класса, указаны их особенности и области применения. Обозначены роевые и эволюционные алгоритмы, подробное описание и результаты реализации и применения которых приведены в работе далее.

1.2. Эволюционные алгоритмы: идея, формализация, основные генетические операторы

В основе работы эволюционных алгоритмов лежат базовые принципы теории биологической эволюции – отбор, мутация, наследование, воспроизводство. Профессором Мичиганского университета в 60-е гг. Д. Холландом был предложен класс эволюционных генетических алгоритмов. В 1975 году вышла книга ученого под названием «Адаптация в естественных и искусственных системах», после которой эти алгоритмы получили всеобщее признание. В работе была заложена дарвиновская теория естественного отбора и селекции сельскохозяйственных культур [5].

Ниже представлены шаги, которые осуществляются при оптимизации задачи классическим генетическим алгоритмом [6].

1. Инициализация исходной популяции хромосом. На первом этапе происходит формирование начальной популяции заданного числа хромосом случайным образом в виде двоичных последовательностей фиксированной длины.

2. Проведение оценки приспособленности хромосом в популяции. Для этого для каждой хромосомы популяции рассчитывается функция приспособленности, которая, как правило, максимизируется.

3. Проверка условия выхода из алгоритма. Критерий выхода из алгоритма определяется исходя из условий решаемой задачи. В том случае, когда известно максимальное значение функции приспособленности условием может быть достижение этого значения с заданной точностью. Другими критериями могут быть истечение заданного времени работы алгоритма или количества итераций. Когда условие выхода из алгоритма выполнено, «наилучшая» хромосома принимается за решение оптимизируемой задачи.

4. Селекция хромосом. На этом этапе происходит выбор родительских особей для очередного поколения. Наибольшей вероятностью попасть в родительский пул обладают хромосомы с самым большим значением функции приспособленности.

5. Применение генетических операторов ведет к формированию новой популяции потомков хромосомами, отобранными на этапе селекции. Основными операторами являются оператор скрещивания и мутации.

6. Формирование новой популяции. Классический генетический алгоритм представляет замену исходной популяции хромосом новой популяцией потомков той же численностью.

7. Выбор лучшей хромосомы на основе целевой функции. В случае выполнения условия остановки алгоритма выводится результат работы алгоритма, которым является хромосома с наибольшим значением функции приспособленности.

В теории эволюции способ передачи признаков родителей потомку играет важную роль при его формировании [7]. Ключевым оператором генетических

алгоритмов является скрещивание, которое определяет их эффективность. На первом этапе скрещивания определяются пары родительских хромосом из пула. Существующие схемы подбора особей в родительскую пару перечислены ниже[8].

1. Панмиксия или случайный способ, при которой вероятность стать родительской особью у всех хромосом одинакова. Этот подход может использоваться в различных классах задач, однако зависимость его эффективности от численности особей в популяции обратная.

2. Инбридинг и аутбридинг – подход, в котором учитывается степень родства хромосом через хеммингово расстояние. Две особи будут считаться близкими родственниками, если хеммингово расстояние между ними меньше заданного положительного значения. Скрещивание генетически похожих особей носит название инбридинга. Если описанное выше условие не выполняется, особи считаются неродственными и их скрещивание называется аутбридингом. Эффективность показывает метод применения инбридинга на ранних этапах работы алгоритма и аутбридинга на поздних.

3. Ассоциативное скрещивание осуществляется путем оценки функции пригодности. Особи подбираются в пару либо по близкому, либо по отличающемуся фенотипу.

Непосредственно сам процесс скрещивания может быть осуществлен по схеме рекомбинации, когда в случае совпадения генов родителей особь получит такой же ген, а в обратном случае с вероятностью 0,5 ген одного из родителей. Другим способом скрещивания родителей является кроссинговер, при котором хромосомы родительских особей разрываются в одном и том же месте и последовательно ими обмениваются.

Еще одним оператором генетических алгоритмов является оператор мутации. Ген в хромосоме при мутации с заданной очень маленькой вероятностью заменяется на противоположный.

Таким образом, в параграфе был рассмотрен один из видов эволюционных алгоритмов – генетический. Было определено, что основной его идеей является

борьба за существование, происходящая между решениями задачи. Каждое из решений называется хромосомой, а все множество решений – популяцией. Был рассмотрен классический генетический алгоритм и его основные шаги. В последующих поколениях появляются новые хорошие и плохие решения задачи, оказывающие влияние на потомство, при этом более удачные решения остаются в новых поколениях с большей вероятностью. Иным подклассом стохастических алгоритмов поиска являются роевые алгоритмы, эффективность которых в решении оптимизационных задач по сравнению с эволюционными исследуется многими авторами.

1.3. Алгоритм роя частиц: история. идея и математическая формализация

Одним из представителей класса популяционных алгоритмов роевого поиска является алгоритм роя частиц. В его основу легло наблюдение за стаей птиц Крейгом Рейнольдсом и создание им в 1986 году программы Boids для имитации их поведения [9].

Поведение птиц в процессе поиска пищи является примером коллективного процесса, так как у большинства видов отсутствует вожак. Птица координирует свое движение относительно положения сородичей, а найдя источник пищи оповещает об этом остальных. Это объясняется тем, что в одиночку птица не может справиться с конкурентами и перемещение в коллективе увеличивает шансы на удачный поиск.

Для имитации процесса ученый запрограммировал поведение каждой птицы в отдельности, а также их взаимодействие. В основе модели лежало три принципа:

1. Каждая птица стремится избежать столкновения с другими.

2. Каждая птица перемещается в том же направлении, что и находящиеся рядом сородичи.

3. Члены стаи пытаются двигаться на одинаковом расстоянии друг от друга.

В результате, полученная поведенческая модель достаточно точно описывала процесс полета стаи, что подтверждалось неоднократными симуляциями в разработанной программе. Рейнольдс К. отмечал, что процесс можно усложнять путем добавления различных факторов: боязни хищников, цели полета и т.д.

Описанные правила, а также работа на схожую тему Ф. Хеппнера и У. Гренандера [10] легли в основу алгоритма роя частиц, который был предложен в 1995 году социальным психологом Джеймсом Кеннеди и инженером-электриком Расселом Эмберхартом [11]. Алгоритм использовался для оптимизации непрерывных нелинейных функций.

На первом этапе каждая частица является случайным решением на области допустимых значений оптимизируемой функции. Далее ей присваивается вектор скорости – значение, на которое следует переместить частицу по области поиска. Это значение обновляется на всех итерациях. Для частицы рассчитывается значение целевой функции. Частица запоминает лучшее решение, которое нашла она и рой за все пройденные итерации.

Обновление вектора скорости реализуется по формуле 1.1 [12]:

$$v_{id}(t+1) = v_{id}(t) + c_1 r_1 (p_{id}(t) - x_{id}(t)) + c_2 r_2 (p_{gd}(t) - x_{id}(t)), \quad (1.1)$$

где v_{id} – компонента *d* частицы *i* на итерации *t;*

$V_i = [v_{i1}, \dots, v_{id}]$ – вектор скорости частицы *i*, где $v_{ij} \epsilon [v_{min}, v_{max}]$;

x_{id} – координата *d* частицы *i* на итерации *t* алгоритма;

$X_i = [x_{i1}, \dots, x_{id}]$ – местоположение частицы *i*, где $x_{ij} \epsilon [x_{min}, x_{max}]$;

p_{id} – координата *d* лучшего решения, найденного частицей;

$P_i = [p_{i1}, \dots, p_{id}]$ – вектор координат, соответствующей лучшему найденному решению частицей *i* на предыдущих итерациях;

p_{gd} – координата *d* лучшего решения, найденного всеми частицами;

$P_g = [p_{g1}, \dots, p_{gd}]$ – вектор координат, соответствующей лучшему найденному решению, найденному частицами на предыдущих итерациях;

r_1, r_2 – случайные числа в интервале (0; 1);

c_1, c_2 – весовые коэффициенты, которые надо подбирать под конкретную задачу.

Для PSO существуют два источника разнообразия, где первый – это разница между текущим лучшим положением и глобально лучшим положением, а второй – разница между текущим положением и исторически лучшим положением частицы. Это отражено на Рисунок *2*.

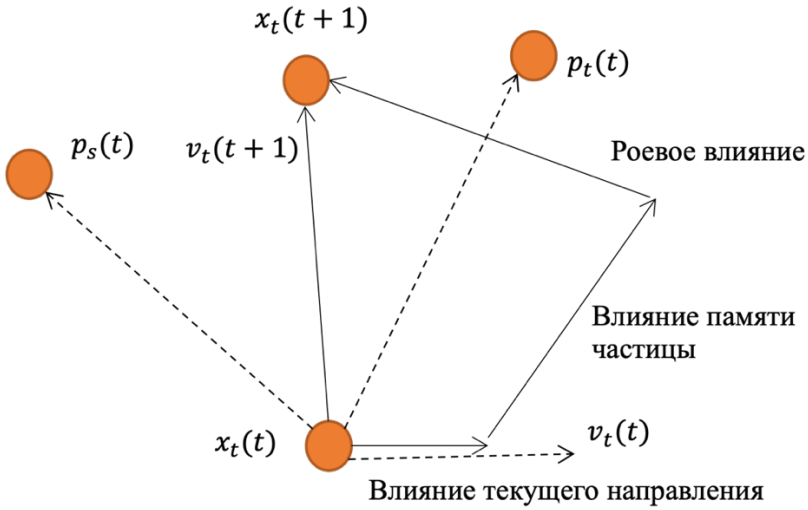


Рисунок 2 - Изменение скорости в алгоритме роя частиц

Формула изменения скорости описывает три основные парадигмы алгоритма роя частиц: распределенное поведение, коллективное управление и локальное взаимодействие с окружающей средой. В качестве первого члена выступает предыдущая скорость (путь инерции), второго когнитивный компонент (память частицы), а третьего социальный компонент (влияние соседства).

После нахождения скорости каждой частицы корректируется ее текущая координата по формуле 1.2:

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1). \quad (1.2)$$

Затем рассчитывается значение целевой функции в каждой новой точке, частица проверяет, не стала ли новая координата лучшей среди всех точек, где она побывала. После среди всех новых точек проверяется, не была ли найдена новая глобально лучшая точка. В случае если такая точка появилась, то запоминаются ее координаты и значение в ней целевой функции.

Существует ряд версий модификаций алгоритма в результате разных способов нахождения скорости частиц. Основная проблема заключается в том, что скорости частиц могут быть сколь угодно большими. Одним из тривиальных способов ограничения скорости частиц является запрет на превышение максимальной скорости. Другие изменения алгоритма описаны ниже.

В 1998 году Юхи Ши и Рассел Эберхарт в своей статье заметили, что одной из главных проблем при решении задач оптимизации является баланс между тщательностью исследования пространства поиска и скоростью нахождения удовлетворительного значения. В зависимости от задачи и характеристик поискового пространства в ней, этот баланс должен быть различным.

С учетом этого, Ши и Эберхарт предложили изменить правило (формула 1.3) обновления векторов скоростей частиц [13]:

$$v_{id}(t+1) = \boldsymbol{w v_{id}}(\mathrm{t}) + c_1 r_1 (p_{id}(t) - x_{id}(t)) + c_2 r_2 (p_{gd}(t) - x_{id}(t)), \quad (1.3)$$

где w - коэффициент инерции, определяет упомянутый баланс между широтой исследования и вниманием к найденным решениям.

Первоначально коэффициент инерции устанавливался как постоянный (например, 0.5), однако результаты показали, что такой вид коэффициента во многих случаях не является сбалансированным. Большой весовой коэффициент облегчает глобальный поиск, а маленький – локальный. В результате было разработано множество вариантов этого коэффициента.

1. Линейные способы корректировки коэффициента инерции

Линейно убывающий коэффициент инерции был так же предложен Юхи Ши и Расселом Эберхартом [14]. В этом подходе коэффициент инерции убывает

от своего наибольшего значения w_{max} до наименьшего w_{min} с увеличением числа итераций по формуле 1.4:

$$w(t) = \frac{t_{max}-t}{t_{max}} (w_{max} - w_{min}) + w_{min}. \quad (1.4)$$

В последующих работах инерционный вес динамически регулировался с помощью механизма увеличения или уменьшения. В частности, в своей работе Харрисон К. [15] увеличивал инерционный вес по формуле 1.5:

$$w(t) = 0.5 * \frac{t}{t_{max}} + 0.4. \quad (1.5)$$

Также есть работы, в которых инерционный вес сначала увеличивался, а затем уменьшался. В этом случае инерционный вес линейно увеличивался с 0.4 до 0.9, а затем снова линейно уменьшался до 0.4 по формуле 1.6.

$$w(t) = \begin{cases} 1 * \frac{t}{t_{max}} + 0.4, & 0 \le \frac{t}{t_{max}} \le 0.5 \\ -1 * \frac{t}{t_{max}} + 1.4, & 0.5 \le \frac{t}{t_{max}} \le 1 \end{cases} \quad (1.6)$$

Существует модификация, в которой учитывается среднее расстояние между частицами в рое (формула 1.7):

$$w(t) = \frac{w_{max} - w_{min}}{D_{max} - D_{min}} * D(t) + \frac{D_{max} w_{min} - D_{min} w_{max}}{D_{max} - D_{min}}, \quad (1.7)$$

где D (t) – среднее расстояние между частицами, $D_{min} \le D(t)\ D_{max}$, $w_{min} \le w(t) \le w_{max}$.

2. Нелинейные способы корректировки коэффициента инерции

В статье Д. Тиана и Дж. Ши [16] собрано 14 различных вариантов нелинейной корректировки коэффициента инерции. Например, в статье Г. Чена и З. Мин [17] исследуется использование экспоненты в изменении коэффициента инерции, формализация представлена в формуле 1.8.

$$w(t) = w_{min} + (w_{max} - w_{min}) * e^{\frac{-t}{t_{max}/10}} \quad (1.8)$$

В статье Б. Джиао и З. Лиань [18] используется формула 1.9:

$$w(t) = w_{start} * u^t, \quad (1.9)$$

где $u \in [1.0001; 1.0005]$.

3. Случайный подход в корректировке коэффициента инерции

Еще одним способом нахождения коэффициента инерции является использование случайного параметра, изменяющегося в каком-либо диапазоне. Например, в формуле 1.10:

$$w(t) = 0.5 + \frac{rand(.)}{2}, \quad (1.10)$$

где rand(.) – случайная величина с нормальным распределением [0;1].

Критерий может не только корректироваться на случайный коэффициент, но и изменяться в зависимости от скорости изменения лучшего найденного решения, как видно из системы 1.11:

$$w(t) = \begin{cases} \alpha_1 + \dfrac{rand\,(.)}{2}, & k \geq 0.05 \\ \alpha_2 + \dfrac{rand\,(.)}{2}, & k < 0.05 \end{cases} \quad (1.11)$$

где $k = \frac{f(t) - f(t-10)}{f(t-10)}$ – скорость изменения оптимального значения;

r – случайная величина с нормальным распределением [0;1];

$\alpha_1 > \alpha_2$.

Помимо инерционного коэффициента можно варьировать и другие параметры, менять значимость опыта отдельной частицы или всего роя через коэффициенты c_1 и c_2, делать перемещение частиц более случайным за счет изменения коэффициентов r_1 и r_2.

Таким образом, в параграфе был рассмотрен алгоритм роя частиц, в основу которого положена концепция роевого интеллекта, а именно поведение стаи птиц во время поиска пищи. При переходе к алгоритму слово «птица» было заменено на «частица». Подход реализуется в дискретном времени и частицы не покидают заданную область поиска. Нахождение оптимального решения

осуществляется всеми особями роя одновременно путем изменения положения частиц на пространстве решений. Это движение осуществляется на основе знаний частицы об общей и своей лучшей позициях, и она стремится к этим положениям. Также движение случайно и обладает инерцией. Ввиду того, что ключевым механизмом работы алгоритма является общение между частицами, изменение структуры обмена информацией может значительно повлиять на результаты. О возможных организациях соседства или топологии частиц рассказано в следующем параграфе.

1.4. Способы адаптации алгоритма роя частиц к задачам оптимизации

В алгоритме роя частиц соседство обеспечивает различные пути связи между его членами и, следовательно, то каким образом рой будет исследовать пространство поиска. Так как топология соседства изменяет схему полета роя, сходимость и разнообразие решений отличаются от топологии к топологии [19].

Весь рой движется вслед за «вожаком» - нашедшей лучшее решение частицей, которая определяется на каждом шаге заново. Каждый раз, когда обнаруживается, что какая-то особь в рое нашла лучшее решение, чем у текущего лидера, она берет лидерство на себя. Лидер может быть глобальным для всего роя или локальным для определенной подгруппы, именуемой соседями. В последнем случае количество лидеров будет равно количеству подгрупп, на которые был разбит рой.

Структура соседства роя определяет способ обмена информацией между частицами. На рисунке 3 представлены наиболее популярные топологии соседства в PSO: полностью связная структура соседства (a), кольцевая окрестность (b), структура окрестности фон Неймана (c), структура соседства четырех кластеров (d). Структура соседства роя определяет способ обмена информацией между частицами [20].

В полностью связной структуре соседства общение между частицами наиболее тесное, в результате рой быстро движется к лучшему найденному

решению и алгоритм быстро сходится. В мультимодальных функциях высока вероятность того, что популяция не выйдет за пределы локального оптимума.

Рой с кольцевой топологией соседства быстро пересекает пространство поиска. Члены роя, организованные в кольцевую структуру, общаются с n непосредственными соседями, n/2 с одной стороны и n/2 с другой (обычно по 2 соседа). Каждая частица инициализируется со специальной меткой, которая не зависит от положения частицы на пространстве решений. Поиск локально лучшего соседа для частицы k осуществляется посредством проверки решений, найденных соседями частицы: k+1, k+2, …, k+n/2, и k-1, k-2, …, k-n/2.

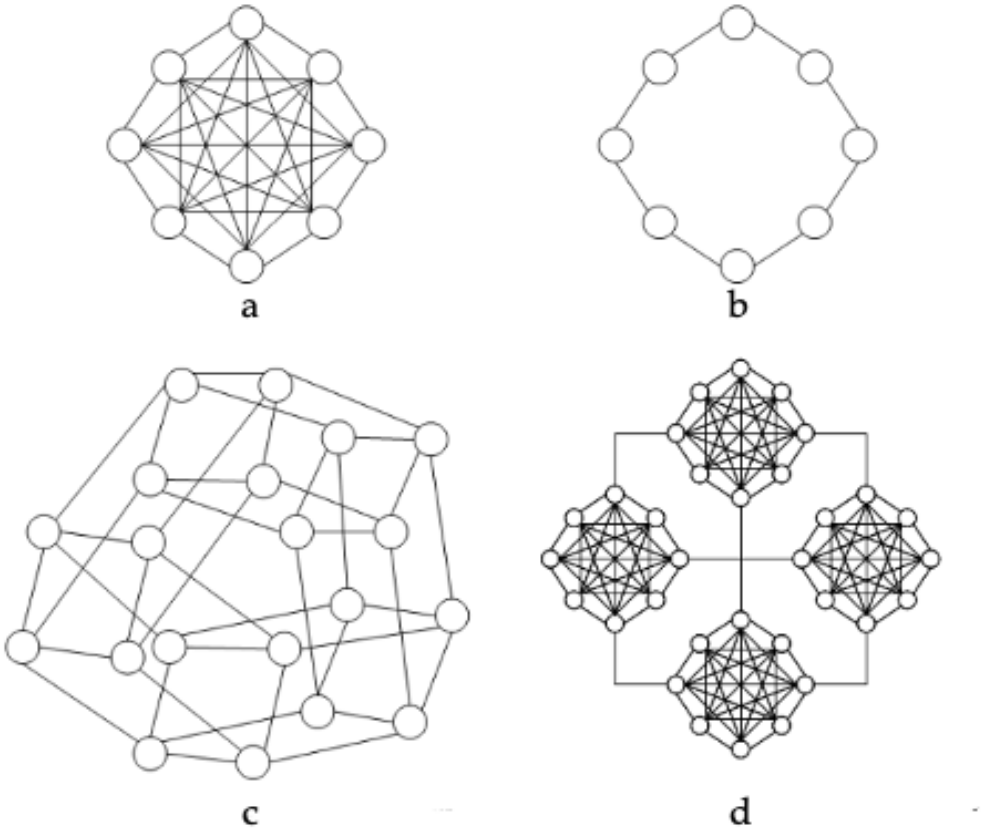


Рисунок 3 – Виды топологии соседства частиц в алгоритме PSO [20]

Существует взаимное притяжение между последовательными частицами. Это можно представить в виде двусвязного списка, как показано на рисунке 4.

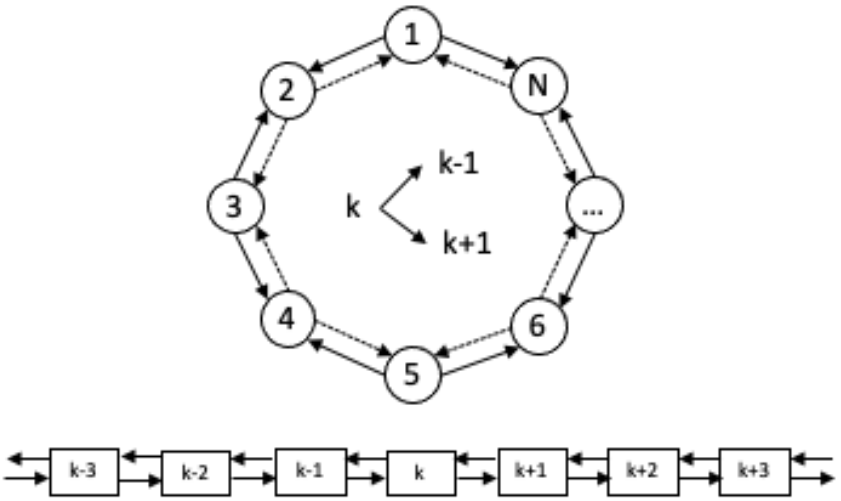


Рисунок 4 – Структура взаимодействия между частицами при использовании кольцевой топологии [20]

Также исследуются другие структуры соседства, Кеннеди и Мендес в своей работе изучают влияние различных структуры соседства частиц на эффективность работы алгоритма PSO. Они рекомендовали структуру фон Неймана, так как она давала более постоянные результаты, чем другие протестированные топологии. На рисунке 5 показана эта топология. Популяция организована в прямоугольную матрицу, и каждая частица связана с особями выше, ниже и по бокам от нее, огибая края.

Структуру фон Неймана так же можно представить в виде двусвязного списка, как показано на рисунке 5. У частицы k есть четыре соседа, частицы k-1 и k+1 по бокам, частицы k-δ и k+δ вверху и внизу, где δ – расстояние, определяемое размером роя. Точно также частицы k-1, k+, k-δ, k+δ имеют частицу k в качестве соседа.

Также следует отметить, что локально лучшее решение в окрестности частицы может быть она сама, так как соседи хуже, чем частица. Однако здесь есть модификация, называемая «selfless model», когда несмотря на то, что лучшее найденное решение соседями хуже, чем лучшее решение, найденное самой частицей, она все равно учитывает только положение соседей.

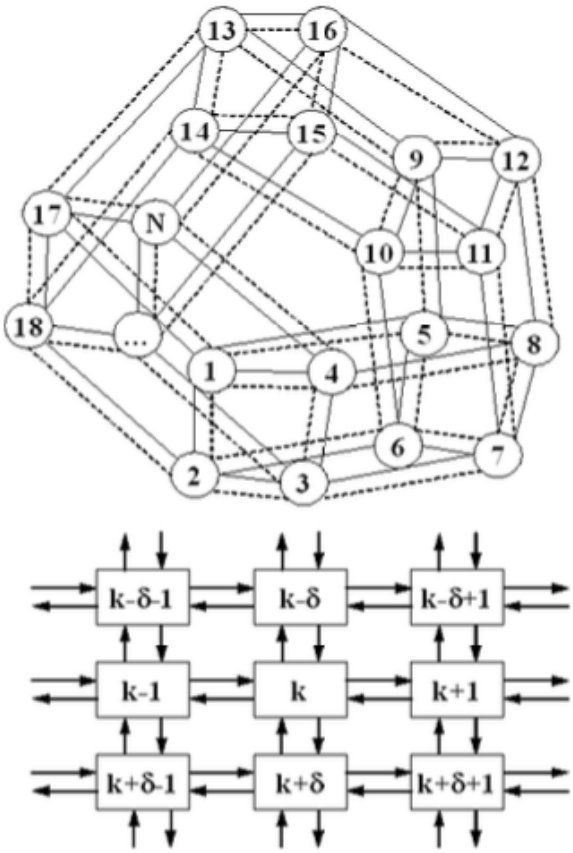


Рисунок 5 – Взаимодействие между частицами при использовании структуры фон Неймана [20]

Существуют модификации, например, перемещают рой по новой кольцевой схеме, называемой односвязным кольцом, что отображено на рисунке 6. Эта структура улучшает результаты эксперимента за счет того, что в качестве соседей частицы k выступают k-2 и k+1, а не k-1 и k+1. В свою очередь у частицы k+1 соседи частицы k-1 и k+2, а у частицы k-1 соседи k-3 и k.

Тогда k притягивает k-1, но k-1 притягивает k только через частицу k+1. Следовательно, частица между ними отменяет взаимное притяжение. Кроме того, информация о лидере передается частицам с меньшей скоростью. Другими словами, для того чтобы всему рою узнать местонахождение лидера нужно больше итераций, чем при кольцевой топологии. Снижая скорость передачи информации односвязное кольцо поддерживает исследование пространства поиска и увеличивает использование лучших решений.

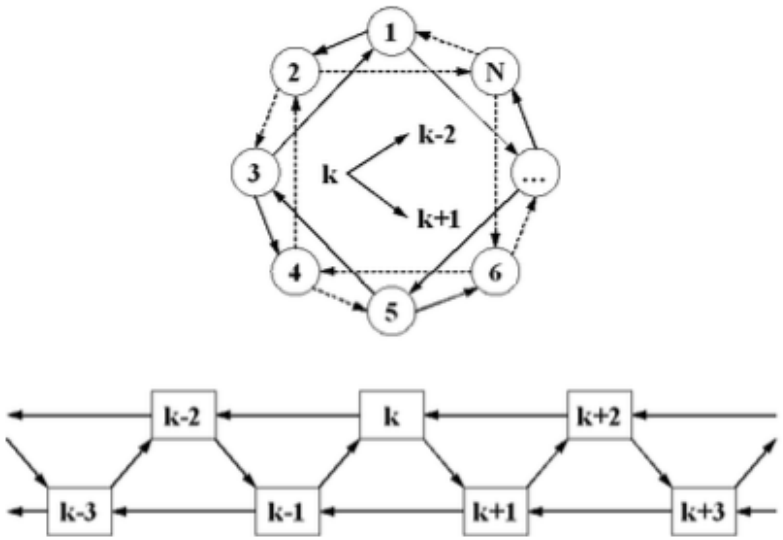


Рисунок 6 – Взаимодействие между частицами при использовании топологии односвязного кольца [20]

Если у односвязной кольцевой топологии у частицы имеется четыре соседа, то каналы связи организованы следующим образом, как показано на рисунке 7.

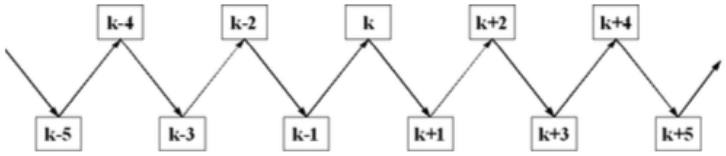
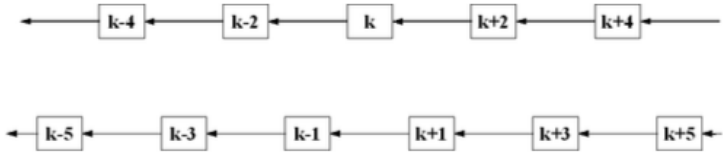
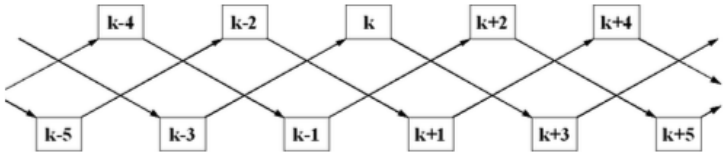
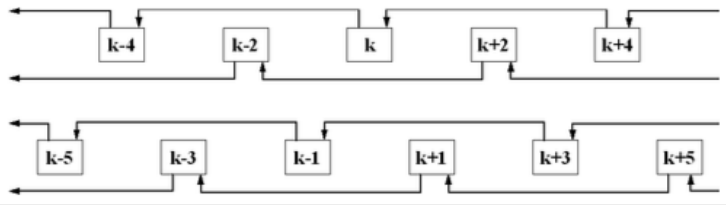
N	Neighbor	Structure
1	$k+1$	
2	$k-2$	
3	$k+3$	
4	$k-4$	

Рисунок 7 – Взаимодействие между частицами при использовании топологии с четырьмя соседями [20]

В результате использования различных топологий можно варьировать интенсивность перемещения роя по пространству поиска. Захватывать более широкие пространства в поиске всех экстремумов, либо уменьшать интервалы перемещения для более глубоко исследования конкретного экстремума.

Процесс поиска может быть разделен на разные фазы: фазу исследования и фазу эксплуатации [21]. Для различных вариантов PSO можно выделить различные этапы поиска, потому что они по-разному работают при решении задач разного типа. Таким образом, чтобы использовать сильные стороны различных вариантов PSO, алгоритм роя частиц может представлять комбинацию нескольких вариантов PSO. Например, глобальная звездная структура (так иначе называют полносвязную топологию алгоритма роя частиц) может использоваться в начале поиска для получения хороших возможностей для исследования, а локальная кольцевая структура может использоваться в конце для более глубоко исследования определенной области. Настройка фаз может быть фиксированной или динамически изменяться во время поиска [22].

Один из вариантов алгоритма роя частиц с несколькими фазами – PSO с фиксированными фазами представлен на рисунке 8. Общая схема работы алгоритма представлена на рисунке. В нем используются две топологии: PSO со звездной структурой (с него алгоритм начинает работу) и PSO с кольцевой структурой (им алгоритм работу заканчивает). Все параметры зафиксированы для каждой фазы поиска, и каждая одна топология сменяет другую за фиксированное количество итераций.

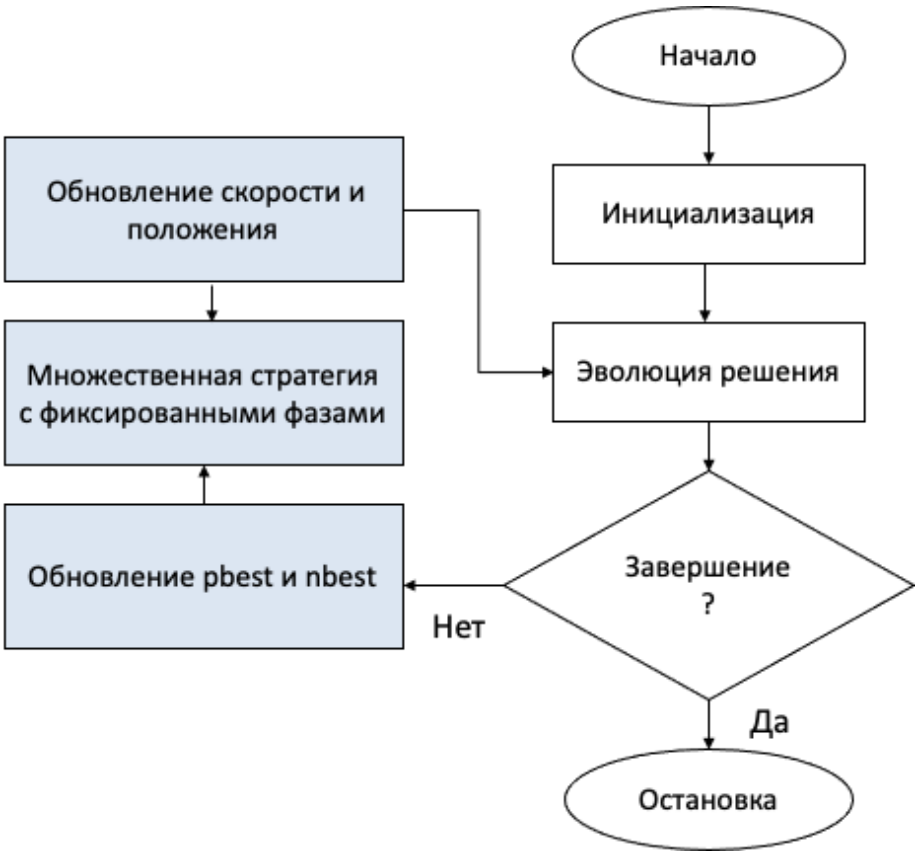


Рисунок 8 – Схема исследования пространства поиска алгоритмом роя частиц с фиксированными фазами

Другой вариант алгоритма роя частиц с несколькими фазами – PSO с динамическими фазами [23]. Как следует из названия, фаза этого алгоритма может динамически изменяться в процессе поиска. Как и алгоритм PSOPH, PSODH начинается со звездообразной структуры и заканчивается PSO с кольцевой структурой. Однако количество итераций, отведенное на каждую из фаз, не фиксируется. Схема данного алгоритма представлена на рисунке 9.

Рисунок 9 – Схема исследования пространства поиска алгоритмом роя частиц с фиксированными фазами

Глобально лучшая позиция (gbest) – это лучшее решение, найденное на данный момент. Если значение gbest не изменилось за k итераций, есть вероятность того, что алгоритм застрял в локальном оптимуме. Вариант PSO будет изменен после того, как значение gbest не будет изменяться определенное количество раз (например, 100).

Таким образом, было выяснено, что топология соседства частиц может в значительной степени повлиять на эффективность работы алгоритма. Например, топология типа клика обеспечивает высокую сходимость алгоритма, но он может остаться в локальном минимуме, а использование кольцевого типа соседства эффективно только для многоэкстремальных функций [24]. В результате использования различных топологий можно варьировать интенсивность перемещения роя по пространству поиска: захватывать более широкие пространства в поиске всех экстремумов, либо уменьшать интервалы перемещения для более глубоко исследования

конкретного экстремума. Кроме того, есть возможность изменять структуру соседства частиц на разных стадиях работы алгоритма, более активно перемещать частицы по пространству поиска на ранних и вступать в фазу эксплуатации локальных экстремумов на поздней стадии.

Таким образом, в главе был проведен обзор особенностей эволюционного отбора и роевого интеллекта, из которого видно, что концепции этих подходов отличаются как на уровне природных механизмов, так и математически. Однако в своих исследованиях некоторые авторы не выделяют роевые алгоритмы в отдельную группу, относя их к эволюционным. Но алгоритмы роевого интеллекта появились на основе концепций роевого интеллекта, в частности алгоритм роя частиц в результате моделирования поведения птиц, а не изучения цепей Маркова.

Уже после распространения этих алгоритмов к ним начали применять математические модели, в том числе цепи Маркова. С помощью них можно доказать сходимость алгоритмов к глобальному оптимуму только при устремлении времени работы к бесконечности, что не объясняет эффективность работы роевых алгоритмов при решении задач с ограничением по времени.

Алгоритмы роевого интеллекта принципиально отличаются от эволюционных, так как не требуют создания новой популяции на каждой итерации через эволюционные механизмы. Они используют коллективные децентрализованные перемещения членов популяции. Был предложен способ отнесения алгоритмов к роевым: в формулах, которые описывают миграцию агентов роя, необходимо наличие объекта, который дает возможность косвенного обмена информацией между ними.

В некоторых работах отмечаются недостатки эволюционных алгоритмов, вызванные их биологическими основами. С. Скиена в своей работе отмечает [1]: «Операции скрещивания и мутации обычно не используют структуры данных, специфичных для данной задачи, вследствие чего большинство переходов дают низкокачественные решения».

Область применения алгоритмов роевого интеллекта помимо задач оптимизации дополняется управлением группами роботов. Данное направление носит название роевой робототехники и активно развивается в последние годы. Управление роботами на основе эволюционных процессов невозможно, так как для этого необходимо было бы организовать создание новых роботов на основе старых непосредственно во время управления.

В следующей главе описаны результаты реализации алгоритмов, основанных на роевых и эволюционных подходах и проанализирована эффективность их работы на примере решения задачи двухуровневой оптимизации.

Глава 2. Применение роевого и эволюционного алгоритма для решения двухуровневой задачи оптимизации

2.1. Математическая постановка задачи двухуровневой оптимизации

Многие оптимизационные проблемы и процессы принятия решения, с которыми сталкиваются государственные и частные организации являются иерархическими в том смысле, что на принятое решение на верхнем уровне оказывают влияние субъекты более низкого уровня. На рисунке 10 представлена общая схема, иллюстрирующая двухуровневую структуру решения проблем, включающую взаимосвязанные задачи оптимизации и принятия решений.

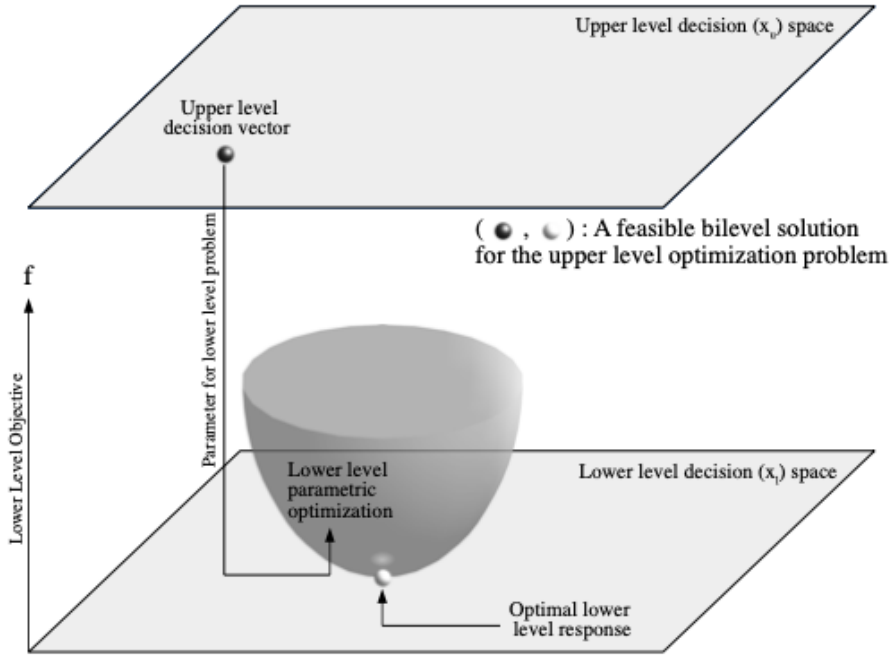


Рисунок 10 – Общая схема задачи двухуровневой оптимизации с существующим решением [25]

Схема иллюстрирует ситуацию, в которой для любого заданного решения верхнего уровня можно найти соответствующий вектор решений на нижнем уровне, который представляет собой оптимальный ответ субъекта нижнего уровня на решения лидера. Вектор решений лидера обозначен через x_u, а вектор решений последователей – x_l. Пара (x_u, x_l^*), где x_l^* – оптимальный ответ на x_u

представляет допустимое решение задачи верхнего уровня, удовлетворяющее ограничениям в виде задачи нижнего уровня. Одной из особенностей двухуровневых задач является их несимметричность с точки зрения двух уровней. Лица, принимающие решения на верхнем уровне обычно полностью осведомлены о проблемах нижнего, в то время как лица на более низких уровнях наблюдают за решениями лидеров и оптимизируют свои собственные стратегии. Нередко цели стремящихся получить прибыль частных лиц вступают в противоречие с интересами контролирующих органов и при решении двухуровневых задач стремятся сократить негативный эффект от действий агентов на экономику или окружающую среду [25].

Задачи двухуровневой оптимизации представляют собой класс сложных оптимизационных задач, в которых одна задача содержит другую в качестве ограничения. Их структура предполагает, что оптимальное решение задачи нижнего уровня – это подходящий кандидат для оптимизационной задачи верхнего уровня. Задача содержит два класса переменных: $x_u \epsilon X_U \subset R^n$ – переменные верхнего уровня и $x_l \epsilon X_L \subset R^m$ – переменные нижнего уровня. Для задачи нижнего уровня оптимизационная задача включает себя x_l, как переменные и x_u, как параметры. Разные x_u образуют различные задачи нижнего уровня, для которых нужно найти оптимальное решение. Оптимизационные задачи верхнего уровня обычно включают в себя все переменные x = (x_u, x_l) и оптимизация проводится для обоих наборов переменных. В статье [26] дана следующая математическая постановка однокритериальной задачи двухуровневой оптимизации: пусть $\psi : R^n \rightrightarrows R^m$ заданное значение отображения, $\psi(x_u) = argmin\{f_0(x_u, x_l): f_j(x_u, x_l) \leq 0, j = 1, \ldots, J\}$, которое описывает ограничение на задачу нижнего уровня, то есть $\psi(x_u) \subset X_L$ для каждого $x_u \epsilon X_U$. Тогда задача двухуровневой оптимизации может быть выражена как общее ограничениие оптимизационной проблемы (2.1):

$$minimize_{x_u \epsilon X_U, x_l \epsilon X_L} F_0(x_u, x_l), \qquad (2.1)$$

при ограничении $x_l \epsilon \psi(x_u),\ F_k(x_u, x_l) \leq 0, k = 1, \ldots, K$

где ψ можно интерпретировать как параметризированное ограничение диапазона возможных решений задачи нижнего уровня x_l

Граф отображения допустимых решений интерпретируется как подмножество $X_U \times X_L$

$$grh\,\bar{\psi} = \{(x_u, x_l)\,\epsilon\, X_U \times X_L \mid x_l\,\epsilon\,\bar{\psi}(x_u)\},$$

которое отображает связи между решениями внешнего уровня и соответствующими оптимальными решениями нижнего уровня. Область определения $\bar{\psi}$, полученная как проекция $grh\,\bar{\psi}$ на пространтсво решений верхнего уровня X_U представляет собой все точки $x_u\,\epsilon\,X_U$, в которых задача нижнего уровня имеет хотя бы одно оптимальное решение, то есть:

$$dom\,\bar{\psi} = \{x_u \mid \bar{\psi}(x_u) \neq \emptyset\}.$$

Точно так же диапазон определяется как:

$$rge\,\bar{\psi} = \{x_l \mid x_l\,\epsilon\,\bar{\psi}(x_u) \text{ для некоторых } x_u\},$$

что соответствует проекции $grh\,\bar{\psi}$ на пространство решений нижнего уровня X_L.

На рисунке 11 показана структура двухуровневой задачи относительно двух компонентов: $grh\,\bar{\psi}$, который дает граф отображения решений нижнего уровня ψ как подмножество $X_U \times X_L$ и график F_0, составленный на $grh\,\bar{\psi}$, который изображает целевую функцию верхнего уровня относительно переменных верхнего уровня, когда нижний уровень оптимален $x_l\,\epsilon\bar{\psi}(x_u)$. Заштрихованная область $grh\,\bar{\psi}$ показывает участки, в которых имеется несколько оптимальных векторов нижнего уровня, соответствующих любому вектору верхнего уровня. С другой стороны, незаштрихованные участки графика демонстрируют области, в которых $\bar{\psi}$ однозначное отображение, то есть существует единственный оптимальный вектор нижнего уровня, соответствующий любому вектору верхнего уровня. Рассматривая $grh\,\bar{\psi}$ как область определения F_0, F_0 можно интерпретировать как функцию от x_u, то есть

$F_0(x_u,\ \psi(x_u))$. Следовательно, когда $\bar{\psi}$ является многозначным, на F_0 появляется заштрихованная область, которая показывает различные значения функции верхнего уровня для любого вектора верхнего уровня с несколькими оптимальными решениями нижнего уровня. Так, на рисунке 22 заштрихованная область $grh\,\bar{\psi}$ соответствует заштрихованной области F_0 для веторов верхнего уровня между $x_u^{(1)}$ и $x_u^{(2)}$.

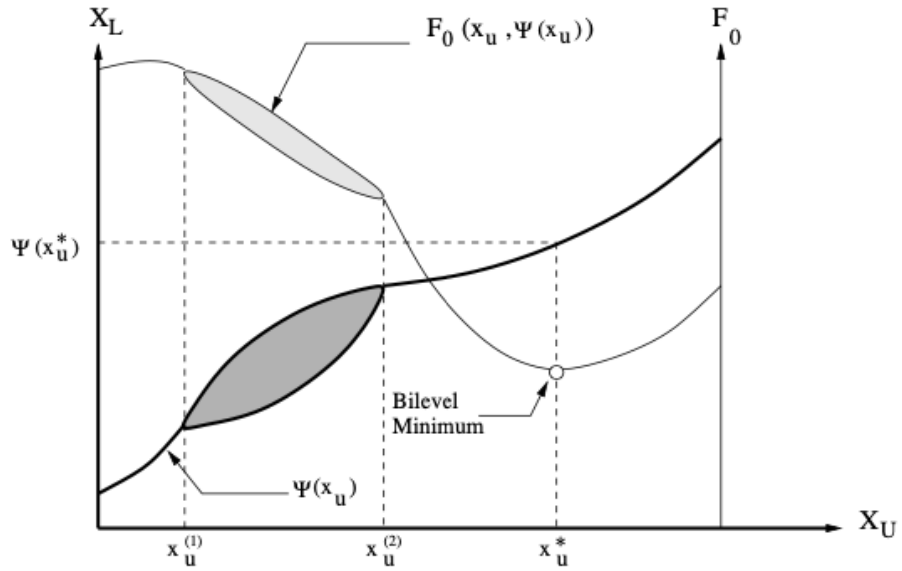


Рисунок 11 – Графическое изображение ψ-соответсвия для x_u и оптимального x_l, и целевая функция верхнего уровня F_0, которая определяет x_u, когда x_l оптимальный [26]

На рисунке 12 изображена более детальная трехмерная иллюстрация ситуации, когда значения целевых функций верхнего и нижнего уровня F_0 и f_0 на различных пространствах решений $X_U\ \times\ X_L$. В данном случае значения F_0 на график относительно переменных решения верхнего уровня аналогичны. Однако в дополнение к предыдущему рисунку описана целевая функция нижнего уровня f_0, зависящая от решения верхнего уровня.

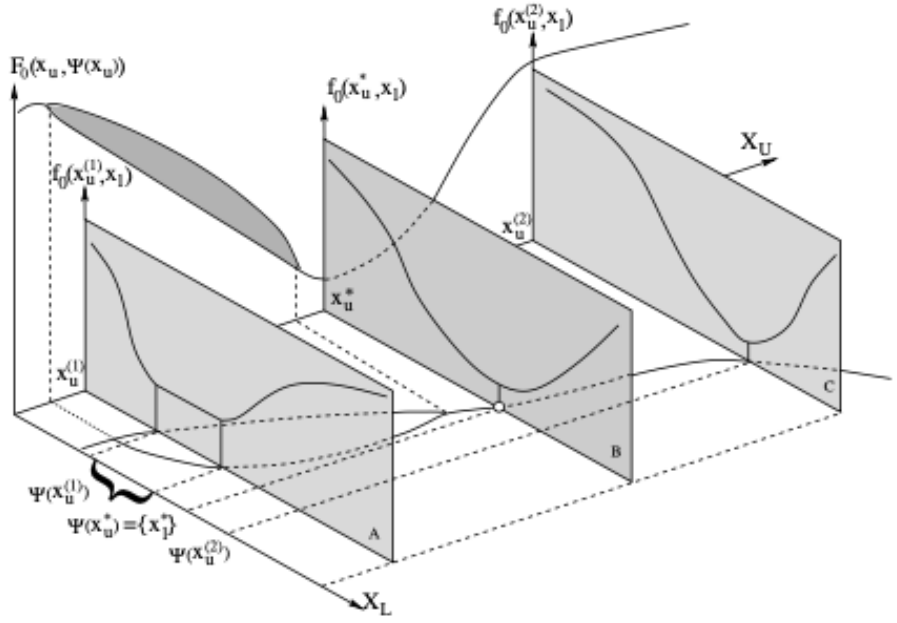


Рисунок 12 – Графическое представление простой задачи двухуровневой оптимизации [26]

На рисунке три области A, B и C представляют три задачи оптимизации более нижнего уровня, параметризованные $x_u^{(1)}$, x^*_u и $x_u^{(2)}$ соответственно. Как только вектор решений верхнего уровня зафиксирован, целевую функцию нижнего уровня можно интерпретировать как функцию от x_l. Следовательно, какждая заштрихованная область показывает график одной переменной f_0 от X_L при заданном зафиксированном x_u. На плоскости A, соответствующей решению верхнего уровня $x_u^{(1)}$, можно увидеть, что на нижнем уровне существует несколько оптимальных решений. Следовательно, ψ также должно быть многозначным в этой точке и набор оптимальных решений нижнего уровня задается как $\psi\left(x_u^{(1)}\right)$. Для двух других областей B и C есть только одно решение нижнего уровня для заданного x_u, что соответствует однозначности ψ в этих точках. Оптимальное решение верхнего уровня находится в точке (x^*_u, x^*_l), где $\psi(x^*_u) = \{x^*_l\}$.

Сравнительный анализ эволюционного и роевого алгоритма проводился для двухуровневой задачи оптимизации, описанной ниже.

Пусть имеется n переменных, $n > 0$. Задана функция двух переменных x_i и c_i (2.2).

$$t_i(x_i, c_i) = t_i^0 \left(1 + 0{,}15 \left[\frac{x_i}{c_i}\right]^4\right), \forall\, i = \overline{1,n} \qquad (2.2)$$

где $t_i^0 > 0$ для всех $i = \overline{1,n}$

Задача двухуровневой оптимизации относительно переменных c = $(c_1, .., c_n)^T$ и x = $(x_1, .., x_n)^T$ представлена в виде выражения (2.3):

$$min_c \sum_{i=1}^{n} t_i(x_i, c_i) x_i \qquad (2.3)$$

при ограничениях на c (2.4):

$$\sum_{i=1}^{n} c_i \leq C, \qquad (2.4)$$
$$c_i \geq 0 \;\forall\, i = \overline{1,n}$$

и x, получаемый как решение оптимизационной задачи нижнего уровня (2.5):

$$x_i = arg\, min_x \sum_{i=1}^{n} \int_0^{x_i} t_i\,(u, c_i)\, du \qquad (2.5)$$

при ограничениях на x (2.6):

$$\sum_{i=1}^{n} x_i = F, \qquad (2.6)$$
$$x_i \geq 0 \;\forall\, i = \overline{1,n}$$

При этом C > 0 и F > 0 – заданные константы.

Таким образом, в параграфе было дано определение и математическая постановка задач двухуровневой оптимизации. Задачи двухуровневой оптимизации представляют собой класс сложных оптимизационных задач, в которых одна задача содержит другую в качестве ограничения. Их структура предполагает, что оптимальное решение задачи нижнего уровня – это подходящий кандидат для оптимизационной задачи верхнего уровня. Дана математическая постановка задачи оптимизации, на которой был проведен сравнительный анализ работы роевого и эволюционного алгоритмов, описанный в следующих параграфах.

2.2. Описание алгоритмов BLEAQ и BPSOQ для решения задачи двухуровневой оптимизации

В качестве эволюционного алгоритма для анализа был выбран bilevel evolutionary algorithm based on quadratic approximation (далее BLEAQ). Он был предложен в статье [26] и показал свою эффективность в решении однокритериальных задач двухуровневой оптимизации.

Оптимизационная стратегия основана на аппроксимации оптимальных переменных нижнего уровня функцией переменных верхнего. На первом шаге инициализируется начальная популяция членов верхнего уровня со случайными переменными верхнего уровня. Для каждого члена верхнего уровня решается задача оптимизации нижнего уровня с использованием схемы оптимизации нижнего уровня и отмечаются оптимальные решения нижнего уровня. На основе полученных оптимальных решений нижнего уровня устанавливается квадратичная зависимость между переменными верхнего уровня и каждой оптимальной переменной нижнего уровня. Путем оценки среднеквадратической ошибки определяется качество полученной модели и, если оно удовлетворительное, ее можно использовать для прогнозирования оптимальных переменных нижнего уровня для любого заданного набора переменных верхнего уровня. Это устраняет необходимость решения задачи оптимизации более низкого уровня. Однако, следует отметить, что даже одно плохое решение может привести к неправильному двухуровнему оптимуму.

На каждой итерации алгоритма квадратичная аппроксимация повторяется и улучшается по мере того, как популяция сходится к истинному оптимуму. По завершении работы алгоритма можно получить не только оптимальные решения, но и приемлемо точные функции, представляющие отношения между переменными верхнего и нижнего уровня в оптимуме. Ниже приведена пошаговая структура работы алгоритма. Иллюстрация работы алгоритма представлена на рисунке 13.

Шаг 1. Инициализация: алгоритм начинается с инициализации случайной популяции размера N, путем генерирования требуемого количества переменных верхнего уровня и проведения процедуры оптимизации функции нижнего уровня путем определения оптимального значения соответствующих переменных нижнего уровня. В качестве оптимизируемой функции выбирается функция верхнего уровня с советующими ограничениями.

Шаг 2. Пометка: все переменные верхнего уровня, которые прошли оптимизацию на нижнем уровне, помечаются единицей, иначе их тэг равен нулю.

Шаг 3. Выбор переменных верхнего уровня: в текущей популяции в качестве одного из родителей выбирается лучшая особь с пометкой 1. Также из популяции случайным образом выбираются $2(\mu - 1)$ члена и проводится соревнование. По соответствующему значению функции верхнего уровня остаются $\mu - 1$ родителей.

Шаг 4. Эволюция на верхнем уровне: особи, отобранные на шаге 2 и 3 становятся родителями. Генерируется λ особей потомства из выбранных μ родителей, используя операторы кроссовера и полиномиальной мутаций.

Шаг 5. Квадратичная аппроксимация: если число особей с тэгом 1 больше чем $\frac{(dim(x_u)+1)(dim(x_u)+2)}{2} + dim(x_u)$, затем выбираются все особи верхнего уровня с тэгом 1, чтобы задать квадратичную функцию, через которую выражается каждая оптимальная переменная нижнего уровня как функция переменных верхнего уровня. Если число переменных с тэгом 1 меньше

$\frac{(dim(x_u)+1)(dim(x_u)+2)}{2} + dim(x_u)$, тогда квадратичная аппроксимация не производится.

Шаг 6. Оптимизация нижнего уровня: если квадратичная аппроксимация проводилась на предыдущем шаге, найти оптимальное значение переменных нижнего уровня для потомства, используя квадратичную аппроксимацию. Если среднеквадратическая ошибка e_{mse} меньше, чем $e_0(1e-3)$, тогда квадратичная аппроксимация считается удачной и потомство помечается тэгом 1, иначе тэг 0. Если квадратичная аппроксимация не производилась на предыдущем шаге, оптимизационный процесс исключения производится для каждого потомка. Если оптимизация неудачная, используем эволюционный алгоритм для решения проблемы.

Шаг 7. Обновление популяции: после нахождения переменных нижнего уровня, r членов выбираются из популяции. Формируется пул из выбранных r членов и λ потомства. Лучшие r членов из пула заменяют r членов из популяции. Проверяется условие завершения выполнения алгоритма.

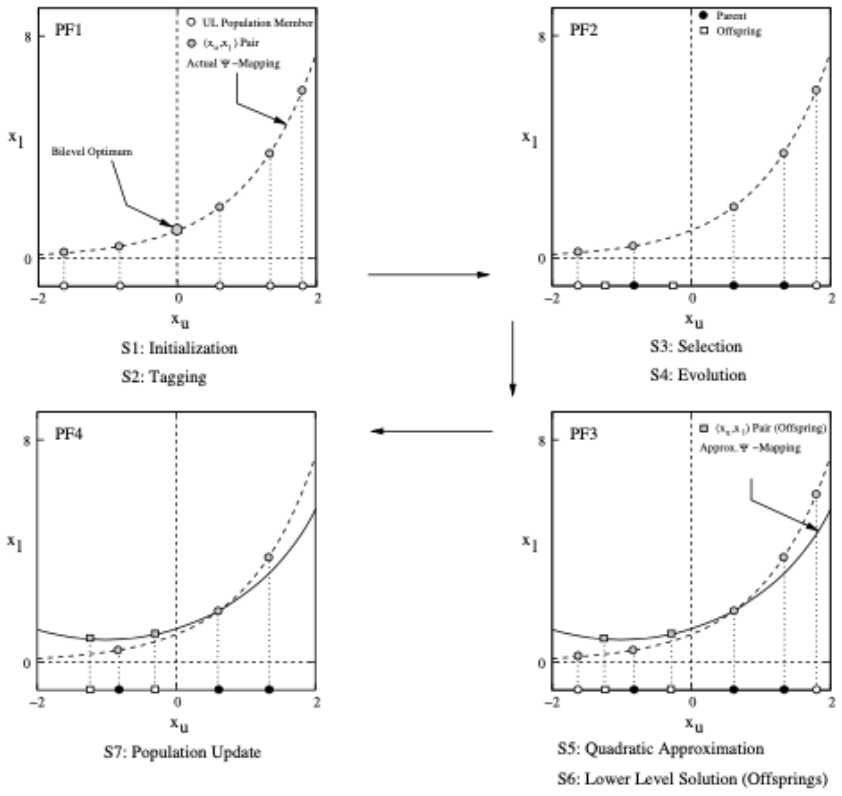


Рисунок 13 – Иллюстрация работы алгоритма BLEAQ [26]

А) Особенности оптимизации на нижнем уровне [27].

На нижнем уровне мы используем квадратичное программирование для нахождения оптимального значения. Если эта процедура была неудачной мы проводим глобальную оптимизацию используя эволюционный подход. Для этого используется целевая функция нижнего уровня с ее ограничениями. Шаги эволюционной оптимизации.

1. Если эволюционная оптимизация нижнего уровня производится без квадратичного программирования, необходимо сгенерировать n решений на нижнем уровне. Если квадратичное программирование уже проводилось, тогда эволюционная оптимизация производится с одним решением, полученным при эволюционной оптимизации и n-1 случайно сгенерированных решений нижнего уровня. Переменные верхнего уровня сохраняются для всех членов популяции.

2. Выбираем 2μ случайных особей из популяции и производим отбор на основе соревнования μ родителей для кроссовера.

3. Лучший родитель среди μ родителей выбирается в качестве index parent и λ членов потомства производятся с помощью операторов кроссовера и мутации.

4. Обновление популяции производится используя r случайных членов популяции. Пул формируется из r выбранных особей и λ потомства, из которого r членов пула заменят r членов популяции.

5. Следующая итерация производится в случае, если не выполнено условие остановки работы алгоритма.

Б) Обработка ограничений.

Задача двухуровневой оптимизации состоит из двух уровней задач оптимизации. На обоих уровнях могут быть ограничения. Однако на верхнем уровне мы включаем ограничения как верхнего, так и нижнего уровня. Это делается для того, чтобы гарантировать решение, которое невозможно реализовать на нижнем уровне, не было допустимым и на верхнем.

Алгоритм использует аналогичную схему обработки ограничений на обоих уровнях, где общее нарушение ограничения для любого решения является

суммированием нарушений всех ограничений равенства и неравенства. Решение $x^{(i)}$ называется «доминирующим при ограничении» над решением $x^{(j)}$, если выполняется любое из следующих условий.

1) Решение $x^{(i)}$ допустимо, а решение $x^{(j)}$ нет.

2) Решение $x^{(i)}$ и $x^{(j)}$ оба неподходящие, но решение $x^{(i)}$ меньше нарушает ограничение.

3) Решение $x^{(i)}$ и $x^{(j)}$ оба подходящие, но целевая функция при $x^{(i)}$ меньше, чем при $x^{(j)}$.

В) Оператор кроссовера

Оператор кроссовера использует трех родителей для создания потомства по следующему правилу (2.7):

$$\mathrm{c} = x^{(p)} + w_\xi d + w_\eta \frac{p^{(2)} - p^{(1)}}{2}, \qquad (2.6)$$

где $x^{(p)}$ – «index parent»;

d = $x^{(p)} - g$, где g - среднее μ родителей;

$p^{(1)}$и $p^{(2)}$ два других родителя;

w_ξ = 0,1 и $w_\eta = \frac{dim(x^{(p)})}{||x^{(p)} - g||}$ – два параметра.

Два параметра w_ξ и w_η описывают степень вариаций в соответствующих направлениях. На верхнем уровне пересечение выполняется только с переменными верхнего уровня, а переменные нижнего уровня определяются из квадратичной функции или вызовом оптимизации нижнего уровня. На нижнем уровне пересечение выполняется только с переменными нижнего уровня, а переменные верхнего уровня остаются фиксированными как параметры.

Данный алгоритм был модифицирован путем изменения способа оптимизации функции верхнего уровня. Вместо эволюционного отбора и получения потомства с помощью кроссовера и полиномиальной мутации было организовано перемещение проинициализированных решений на верхнем уровне по пространству поиска. Для этого каждое решение верхнего уровня было

представлено как частица, обладающая памятью. Частица знает свое лучшее положение и лучшее положение, найденное роем за все итерации.

На шаге 3 после отбора агентов на верхнем уровне происходит их перемещение по пространству поиска, причем агент ориентируется на свое лучшее положение, а также на лучшее положение, найденное всем роем и их скорость обновляется по формуле 1.3. Лучшим считается положение, в котором было получено наименьшее значение целевой функции верхнего уровня. После для частиц с обновленным положением вычисляется значение переменных нижнего уровня из уравнения, полученного квадратичной аппроксимацией. Если среднеквадратическая ошибка e_{mse} меньше чем $e_0(1e-3)$, то квадратичная аппроксимация считается удачной и значения переменных нижнего уровня ставятся в соответствие переменным верхнего уровня. Иначе, для каждого вектора переменных верхнего уровня переменные нижнего уровня вычисляются точным способом, путем минимизации целевой функции нижнего уровня.

Таким образом, в параграфе было дано описание эволюционного и роевого алгоритма, описание реализации и сравнительный анализ которых приведены в следующем параграфе. В качестве эволюционного алгоритма был выбран алгоритм BLEAQ, показавший свою эффективность в решении задач однокритериальной двухуровневой оптимизации. Этот алгоритм был модифицирован путем замены эволюционных механизмов в нем роевыми эвристиками и новый алгоритм BPSOQ.

2.3. Реализация алгоритмов BLEAQ и BPSOQ и их сравнительный анализ

Описанные в параграфе 2.2 алгоритмы были реализованы в среде Wolfram Mathematica 13.0. Алгоритм проверялся для функции с тремя переменными на верхнем и тремя переменными на нижнем уровнях. Константы C и F, через которые определяется ограничения задачи были приняты равными 130 и 100. Численность роя в алгоритме BPSOQ и популяции в алгоритме BLEAQ равняется 50. В обоих алгоритмах было реализовано 500 итераций.

Анализ алгоритма BPSOQ

В алгоритме значения коэффициентов w, c_1 и c_2 равны 0,5, 0,3 и 0,2 соответственно, r_1 и r_2 на каждой итерации принимают случайное значение на интервале (0;1). О каждом агенте роя собрана в виде информация в виде датасета со следующими столбцами:

1. Tag – тэг агента (значение 0 или 1).
2. UpperVariables – значения переменных верхнего уровня.
3. LowerVariables – значения переменных нижнего уровня.
4. UpperLevelFunctionValue – значение целевой функции верхнего уровня.
5. LowerLevelFunctionValue – значение целевой функции нижнего уровня.
6. BestParticlesValue – лучшее значение целевой функции верхнего уровня, полученное частицей за все итерации.
7. BestParticlesPosition – лучшие значения переменных верхнего уровня, найденные частицей за все итерации.
8. Velocity – скорость частицы.

Лучшее значение целевой функции и соответствующие ему значения переменных верхнего уровня, найденные роем за все итерации, хранятся в переменных bestValue и bestPosition.

На первом шаге происходит инициализация переменных верхнего и нижнего уровня случайным образом при условии выполнения ограничений 2.4 и 2.6. Затем происходит пересчет значений целевых функций верхнего и нижнего уровней, обновление лучших позиций у каждой отдельной частицы, а также лучшего значения целевой функции верхнего уровня и лучшего положения переменных верхнего уровня для всего роя.

После этого происходит пересчет скорости и положения частиц, а также проверка выполнения ограничений на переменные верхнего уровня 2.4. Реализация проверки ограничений представлена на рисунке 14.

```
For[i = 1, i ≤ Length@chosenParentsVector, i++,
 For[j = 1, j ≤ n, j++,
  newPos = parentsDataset[chosenParentsVector[i]]["UpperVariables"][j] + offspringVelocity[i, j];
  If[newPos < 0,
   newPos = 0.001
  ];
  upperOffspringVector[i, j] = newPos
 ]
];

 For[i = 1, i ≤ Length@chosenParentsVector, i++,
  sum = Total@upperOffspringVector[i];
  If[sum > C,
   {
    dif = sum - C,
    upperOffspringVector[i] = upperOffspringVector[i] - upperOffspringVector[i] / sum * dif
   }
  ]
 ];
```

Рисунок 14 – Проверка выполнения ограничения 2.4 у переменных верхнего уровня

Далее происходит оптимизация переменных нижнего уровня. Сначала происходит отбор переменных с тэгом и затем с помощью функции NonlinearModelFit происходит аппроксимирование переменных нижнего уровня, как функции от переменных верхнего уровня. Затем вычисляется среднеквадратическая ошибка и в случае, если аппроксимация прошла удачно, для новых переменных верхнего уровня вычисляются переменные нижнего. Реализация этого процесса представлена на рисунке 15.

```
keysOfTaggedVariables = Keys[Select[(Transpose@parentsDataset)["Tag"], # == 1 &]]

  data = {xValues, yValues};
  polynom0 = NonlinearModelFit[Transpose@data, a0 + b0 * x + c0 * (x ^ 2), {a0, b0, c0}, x];
  polynom = Append[polynom, polynom0];
 ];
 quadraticApproximationIndicator = 1;
}
];
If[quadraticApproximationIndicator == 1,
 approximationError = polynom[#]["ANOVATableMeanSquares"][2] & /@ Range[Length@polynom];,
 approximationError = {0};
];
```

Рисунок 15 – Проверка тэга и реализация квадратичной аппроксимации переменных нижнего уровня, как функции от переменных верхнего уровня

В случае, если аппроксимация оказалась неудачной, параметризируемая переменными верхнего уровня функция нижнего уровня оптимизируется

точным методом. Затем происходит обновление информации в датасете (рисунок 16) и обновление лучших значений, найденных всем роем и каждой частицей.

```
For[k = 1, k ≤ λ, k++,
 parentsDataset = Insert[parentsDataset,
   chosenMembersOfParentPopulation[[k]] → Association[
     "Tag" → 1,
     "UpperVariables" → pool1[[5 + positionOfBestOffspring[[k]]]],
     "LowerVariables" → pool1[[positionOfBestOffspring[[k]]]],
     "UpperLevelFunctionValue" → upperLevelFunctionValue[[k]],
     "LowerLevelFunctionValue" → lowerLevelFunctionValue[[positionOfBestOffspring[[k]]]],
     "BestParticlesValue" → bestOffspringValue[[k]],
     "BestParticlesPosition" → bestOffspringPosition[[k]],
     "Velocity" → offspringVelocity[[positionOfBestOffspring[[k]]]]
    ],
   Key[chosenMembersOfParentPopulation[[k]]]
  ]
];
```

Рисунок 16 – Обновление положения частиц в датасете

На рисунке 17 представлена информация об агентах роя после проведения 500 итераций работы алгоритма. Лучшее значение показала частица под номером 46: лучшее значение целевой функции верхнего уровня в ней получилось равным 3,06332, а целевое значение функции нижнего уровня – 101,146. Соответствующие им значения переменных верхнего и нижнего уровня: (0,001; 1,30762; 127,186), (0,0000792817; 0,0000792817; 99,9998). Как видно из рисунка, лучшие решения для двухуровневой задачи отличаются от лучших решений задачи верхнего уровня, которыми являются частицы. Это объясняется тем, что частица меняет свое значение только в том случае, если получено лучшее значение целевой функции на верхнем уровне.

	Tag	UpperVariables	LowerVariables	UpperLevelFunctionValue	LowerLevelFunctionValue	BestParticlesValue	BestParticlesPosition	Velocity
31	1	{0.001, 1.35301, 124.394}	{0.0000792817, 0.0000792817, 99.9998}	3.07029	101.253	3.0606	{0.926938, 0.001, 125.213}	{−0.182949, 0.0871475, 1.17013}
32	1	{2.6462, 4.71182, 125.17}	{0.0000792817, 0.0000792817, 99.9998}	3.07879	101.222	3.06507	{0.001, 3.23781, 121.765}	{1.49167, −0.378388, 0.803566}
33	1	{0.001, 1.30762, 127.186}	{0.0000792817, 0.0000792817, 99.9998}	3.0639	101.146	3.06208	{1.58465, 1.6274, 124.579}	{−0.72589, −0.938686, 1.06294}
34	1	{1.58465, 1.6274, 124.579}	{0.0000792817, 0.0000792817, 99.9998}	3.0628	101.246	3.05929	{1.75609, 3.33031, 127.364}	{1.13799, −3.44204, 1.26347}
35	1	{0.001, 1.35301, 124.394}	{0.0000792817, 0.0000792817, 99.9998}	3.06272	101.253	3.06228	{0.001, 1.35301, 124.394}	{−0.182949, 0.0871475, 1.17013}
36	1	{0.526389, 2.1689, 125.046}	{0.0000792817, 0.0000792817, 99.9998}	3.0593	101.227	3.0593	{0.526389, 2.1689, 125.046}	{0.00315046, −2.26992, −1.21734}
37	1	{0.530751, 0.001, 126.279}	{0.0000792817, 0.0000792817, 99.9998}	3.07294	101.18	3.06461	{0.751357, 1.80313, 124.817}	{0.529751, −2.75548, −0.906518}
38	1	{2.61652, 5.01607, 125.479}	{0.0000792817, 0.0000792817, 99.9998}	3.06427	101.21	3.06375	{2.17833, 2.7036, 125.849}	{1.90419, 1.54062, 0.04061}
39	1	{2.61652, 5.01607, 125.479}	{0.0000792817, 0.0000792817, 99.9998}	3.0611	101.21	3.0611	{2.61652, 5.01607, 125.479}	{1.90419, 1.54062, 0.04061}
40	1	{1.3364, 1.75309, 124.677}	{0.0000792817, 0.0000792817, 99.9998}	3.06835	101.242	3.0606	{1.15453, 5.0902, 124.366}	{1.3354, 0.400082, 0.283383}
41	1	{0.001, 2.24631, 126.123}	{0.0000792817, 0.0000792817, 99.9998}	3.0638	101.186	3.06059	{0.444534, 5.82048, 123.829}	{−1.12882, 0.443173, 1.30611}
42	1	{0.001, 2.78176, 124.426}	{0.0000792817, 0.0000792817, 99.9998}	3.06425	101.252	3.06343	{0.001, 2.78176, 124.426}	{−1.1705, −0.630968, 0.416971}
43	1	{0.001, 0.200787, 124.633}	{0.0000792817, 0.0000792817, 99.9998}	3.0598	101.243	3.05462	{0.001, 0.200787, 124.633}	{−1.13883, −1.15222, 0.238722}
44	1	{0.523239, 4.43882, 126.263}	{0.0000792817, 0.0000792817, 99.9998}	3.05462	101.18	3.05462	{0.523239, 4.43882, 126.263}	{−0.63129, −0.651383, 1.89676}
45	1	{3.61955, 1.24415, 128.733}	{0.0000792817, 0.0000792817, 99.9998}	3.45	101.092	3.0639	{4.55856, 6.58699, 123.475}	{1.86346, −2.08615, 1.36883}
46	1	{0.001, 1.30762, 127.186}	{0.0000792817, 0.0000792817, 99.9998}	3.06332	101.146	3.06111	{2.86025, 0.001, 123.47}	{−0.72589, −0.938686, 1.06294}
47	1	{0.751357, 1.80313, 124.817}	{0.0000792817, 0.0000792817, 99.9998}	3.06606	101.236	3.06373	{0.306345, 5.692, 124.641}	{−0.293051, −1.11812, 0.749494}
48	1	{3.86077, 1.28658, 125.205}	{0.0000792817, 0.0000792817, 99.9998}	3.0611	101.221	3.0611	{3.86077, 1.28658, 125.205}	{0.241219, 0.0424298, −3.52804}
49	1	{0.241845, 1.41997, 125.001}	{0.0000792817, 0.0000792817, 99.9998}	3.07226	101.229	3.0618	{0.001, 2.01922, 122.563}	{0.240845, 0.913076, −0.0521368}
50	1	{0.001, 2.50131, 124.987}	{0.0000792817, 0.0000792817, 99.9998}	3.05733	101.229	3.05733	{0.001, 2.50131, 124.987}	{−1.55055, 1.19369, −2.19858}

rows 31–50 of 50

Рисунок 17 – Информация о состоянии агентов роя №31–50 на 500 итерации работы алгоритма

На рисунках 18 и 19 показано изменение лучших найденных значений целевых функций верхнего и нижнего уровней соответственно. За 500 итераций целевая функция верхнего уровня уменьшилась с 3,18873 до 3,06332, а целевая функция нижнего уровня уменьшилась с 107,4 до 101,146. Изменение целевых функций происходило неравномерно, причем несмотря на замедление изменения лучшего найденного значения с увеличением количества итераций, дальнейшее продолжение оптимизации имело смысл.

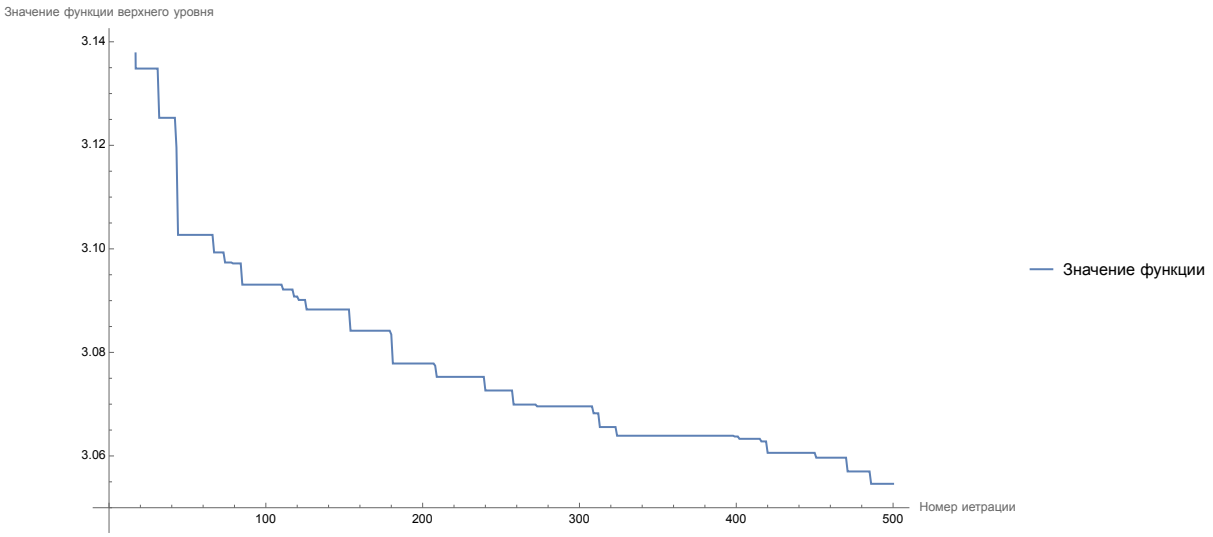


Рисунок 18 – Изменение лучшего найденного значения целевой функции верхнего уровня за 500 итераций

Наличие случайной составляющей при перемещении частиц по пространству поиска обеспечивает подвижность роя и как следствие более тщательный поиск в локальных минимумах.

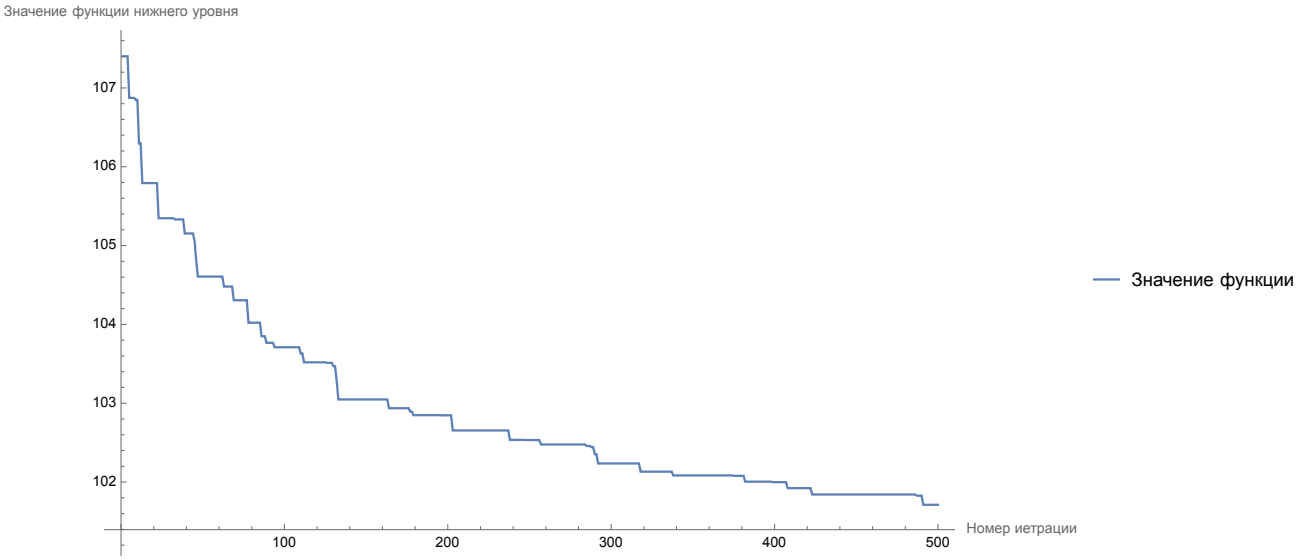


Рисунок 19 – Изменение лучшего найденного значения целевой функции нижнего уровня за 500 итераций

На рисунке 19 изображены лучшие значения целевой функции верхнего уровня, найденные роем за 500 итераций. Полученные значения находятся на достаточно большом интервале относительно общего изменения лучшего найденного значения целевой функции верхнего уровня. В окрестность входят значения от 3,06332 до 3,08245. Увеличение итераций в алгоритме может сократить этот диапазон, однако количество агентов подобрано удачно, так как

найден баланс между временем работы алгоритма и разнообразием имеющихся решений. Такое количество частиц позволяет менять топологию их соседства, что также влияет на направление и скорость перемещения частиц по пространству поиска.

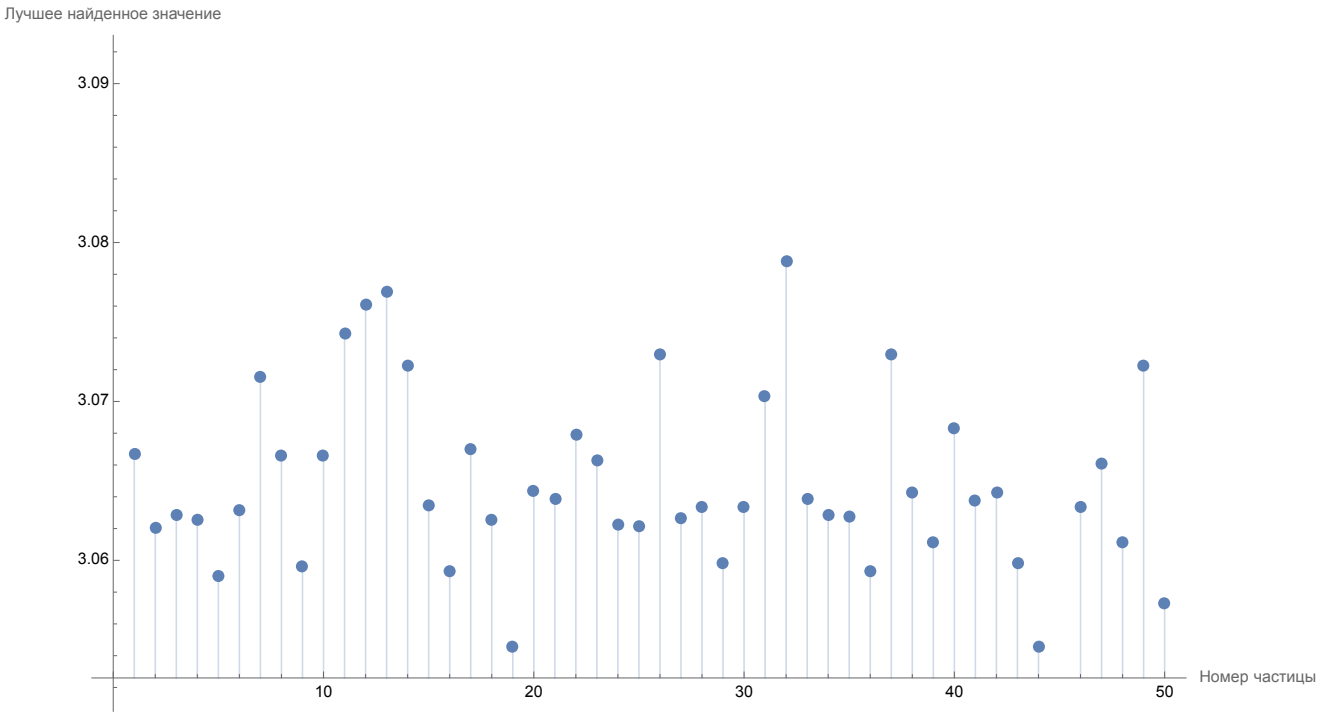


Рисунок 19 – Лучшее значение целевой функции верхнего уровня, найденное каждой частицей за 500 итераций

Анализ алгоритма BLEAQ

В алгоритме были приняты следующие значения коэффициентов $w_1 = 0{,}1;\ r = 2;\ \mu = 3;\ \lambda = 2;\ p = 2$. Вероятности кроссовера и мутации равны 0,5 и 0,1 соответственно. О каждом агенте роя собрана в виде информация в виде датасета со следующими столбцами:

1. Tag – тэг агента (значение 0 или 1).
2. UpperVariables – значения переменных верхнего уровня.
3. LowerVariables – значения переменных нижнего уровня.
4. UpperLevelFunctionValue – значение целевой функции верхнего уровня.
5. LowerLevelFunctionValue – значение целевой функции нижнего уровня.

Этап оптимизации функции нижнего уровня аналогичен алгоритму BPSOQ, изменяется подход к оптимизации функции верхнего уровня. После инициализации решений случайным образом в соответствии с ограничениями задачи происходит отбор родительских особей для создания следующего поколения. Реализация этого процесса показана на рисунке 20. Родители в количестве $\mu - 1$ отбираются случайным образом. Последняя родительская особь – это лучшее решение функции верхнего уровня, найденное на предыдущих итерациях.

```
keysForTournament = RandomSample[parentsDatasetKeysList, 2 * (μ - 1)];
bestKeysFromTournament = TakeSmallest[(Transpose@KeyTake[parentsDataset, keysForTournament])["UpperLevelFunctionValue"], 2];

chosenParentsVector = Append[Keys@bestKeysFromTournament, bestUpperLevelMember[1]];
}
```

Рисунок 20 – Отбор родительских особей из популяции

Далее реализуются эволюционные механизмы – кроссовер и мутация (рисунок 21). Потомки подвергаются кроссоверу в случае если случайное число, сгенерированное на интервале [0;1] с помощью функции RandomReal, больше его вероятности. Находится среднее значение отобранных родительский особей для каждой переменной и вычисляется оператор кроссовера. Далее происходит проверка выполнения ограничений у полученных особей. Таким же образом в случае, если случайное число, сгенерированное на интервале [0;1] с помощью функции RandomReal, больше вероятности мутации, реализуется этот механизм и после него так же происходит проверка ограничений.

После оптимизации переменных нижнего уровня относительно полученных переменных верхнего уровня происходит эволюционное соревнование потомства со случайно выбранными особями из популяции. Победители занимают места в популяции, происходит обновление лучших значений и алгоритм возвращается к шагу два либо заканчивает свою работу.

```
For[
 i = 1, i ≤ λ, i++,
 crossoverNumber = RandomReal[];
 If[crossoverNumber ≤ crossoverProbability,
  {
   upperOffspringVariable = {},
   For[
    k = 1, k ≤ n, k++,
    {
     g = Mean[
       {
        taggedParentsDataset[chosenParentsVector[1]]["UpperVariables"][k],
        taggedParentsDataset[chosenParentsVector[2]]["UpperVariables"][k],
        taggedParentsDataset[chosenParentsVector[3]]["UpperVariables"][k]
       }
      ],
     d = taggedParentsDataset[chosenParentsVector[3]]["UpperVariables"][k] - g,
     If[d ≠ 0,
      w2 = variableNumberC / Abs[d],
      w2 = 0.1
     ],
     crossoverOperator = Abs[
       taggedParentsDataset[chosenParentsVector[3]]["UpperVariables"][k]  + w1 * d +
        w2 * ((taggedParentsDataset[chosenParentsVector[1]]["UpperVariables"][k] - taggedParentsDataset[chosenParentsVector[2]]["UpperVariables"][k]) / 2)
      ],
     If[
      crossoverOperator ≤  0,
      crossoverOperator = 0.0001
     ],
     upperOffspringVariable = Append[upperOffspringVariable, crossoverOperator ]
```

Рисунок 21 – Реализация кроссовера для созданного потомства

На рисунке 22 представлена информация об агентах роя после проведения 500 итераций работы алгоритма BLEAQ. Примечательно, что после завершения работы алгоритма все 50 особей популяции одинаковые. Лучшее найденное значение функции верхнего уровня – 3,15845, нижнего – 103,169. Соответственно лучшее решение функции верхнего уровня – (0,0755244; 1,28539; 98,6391), нижнего – (0,0000786046; 0,0000792792; 99,9998).

	Tag	UpperVariables	LowerVariables	UpperLevelFunctionValue	LowerLevelFunctionValue
1	1	{0.0755244, 1.28539, 98.6391}	{0.0000786046, 0.0000792792, 99.9998}	3.15845	103.169
2	1	{0.0755244, 1.28539, 98.6391}	{0.0000786046, 0.0000792553, 99.9998}	3.15845	103.169
3	1	{0.0755244, 1.28539, 98.6391}	{0.0000786046, 0.0000792809, 99.9998}	3.15845	103.169
4	1	{0.0755244, 1.28539, 98.6391}	{0.0000786046, 0.0000792553, 99.9998}	3.15845	103.169
5	1	{0.0755244, 1.28539, 98.6391}	{0.0000786046, 0.0000792817, 99.9998}	3.15845	103.169
6	1	{0.0755244, 1.28539, 98.6391}	{0.0000786046, 0.0000792566, 99.9998}	3.15845	103.169

Рисунок 22 – Информация о состоянии агентов популяции №1–6 на 500 итерации работы алгоритма

На рисунках 23 и 24 показано изменение значения функций верхнего и нижнего уровня соответственно только для 100 итераций, так как изменение этих значений перестало происходить на 49 итерации. В ходе работы алгоритма

лучшее значение целевой функции верхнего уровня уменьшилось с 3,19342 до 3,15845, а целевой функции нижнего – с 109,672 до 103,169.

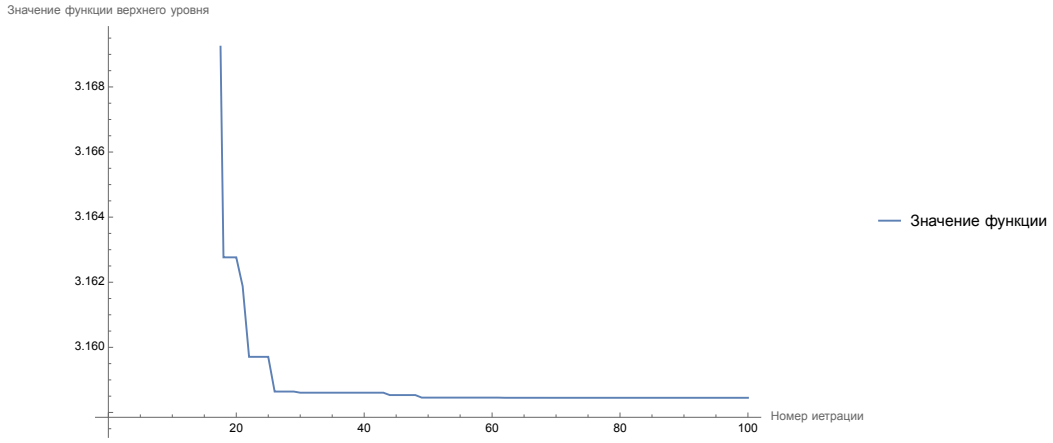


Рисунок 23 – Изменение лучшего найденного значения целевой функции верхнего уровня за 100 итераций

Это можно объяснить тем, что двумя потомками, участвующими в соревновании на место в популяции являются две ее случайные особи. В какой-то момент потомки оказываются хуже своих родителей в результате чего обновление популяции прекращается.

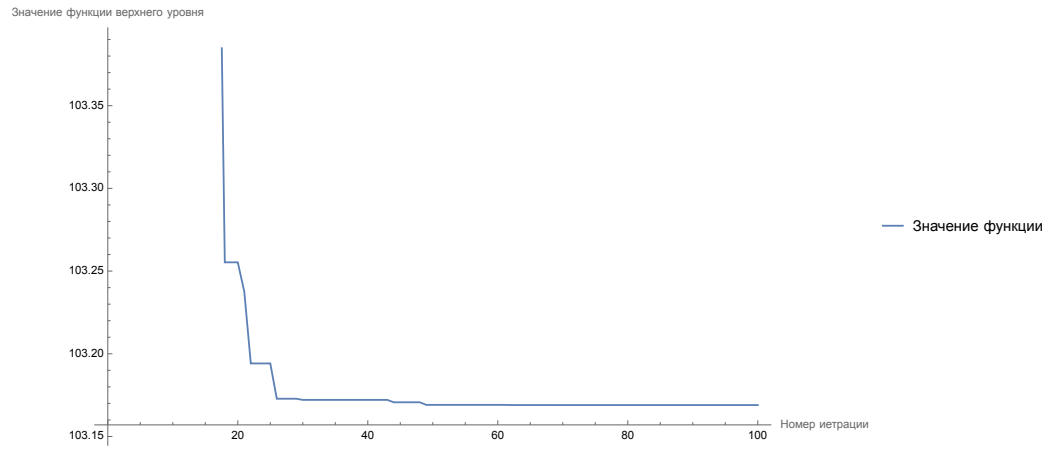


Рисунок 24 – Изменение лучшего найденного значения целевой функции нижнего уровня за 100 итераций

На рисунке 25 показана динамика изменения лучшего значения целевой функции верхнего уровня, найденного алгоритмами BLEAQ и BPSOQ за 500 итераций. Алгоритм с применением роевых эвристик оказался более эффективным с точки зрения лучшего найденного значения, исследования пространства решений.

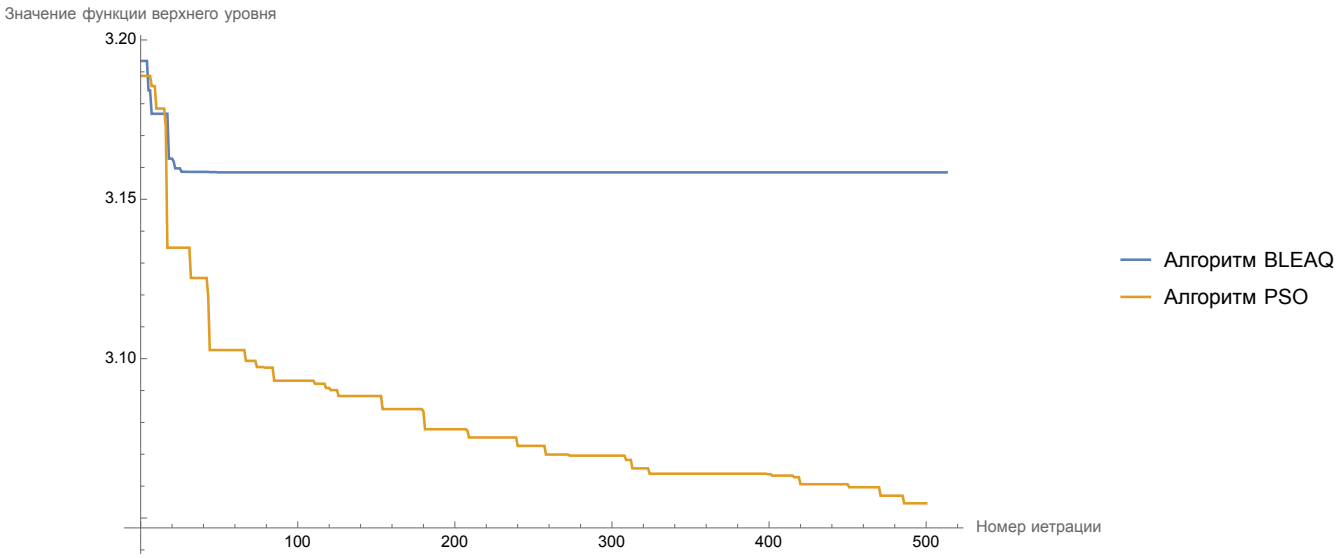


Рисунок 25 – Динамика изменения лучшего значения целевой функции верхнего уровня, найденного алгоритмами BLEAQ и BPSOQ за 500 итераций

В таблице 1 собраны основные результаты применения алгоритмов BLEAQ и BPSOQ для решения задач двухуровневой оптимизации.

Таблица 1. Результаты работы алгоритмов BLEAQ и BPSOQ

	BLEAQ	**BPSOQ**
Количество агентов	50	50
Лучшее значение целевой функции верхнего уровня	3,15845	3,06332
Лучшее значение целевой функции нижнего уровня	103,169	101,146
Время работы функции	497,766	523,655

Несмотря на то, что алгоритм BPSOQ оказался медленнее, этой разницей во времени работы можно пренебречь для получения значительно более точного результата.

Таким образом, в результате проведения сравнительного анализа алгоритмов BLEAQ и BPSOQ для решения двухуровневых задач оптимизации на тестовых примерах более эффективно показали себя роевые эвристики. В первую очередь это объясняется что в эволюционном алгоритме потомками, участвующими в соревновании на место в популяции являются две ее случайные особи. В какой-то момент потомки оказываются хуже своих родителей в результате чего обновление популяции прекращается. Кроме того, алгоритм

роя частиц реализован на идее коллективного поведения и общение всех участников популяции между собой позволяет более тщательно исследовать пространство поиска, захватывая большие расстояния в начале работы алгоритма и более глубоко исследуя локальные минимумы в конце. Для двухуровневой задачи оптимизации топологии транспортной сети было решено использовать именно роевой интеллект, подобрав перед этим наиболее эффективную адаптацию. Код реализованных алгоритмов расположен по ссылке [28].

Глава 3. Решение двухуровневой задачи оптимизации топологии транспортной сети алгоритмом PSODP – leBlanc с динамическим коэффициентом инерции

3.1. Математическая постановка двухуровневой задачи оптимизации топологии транспортной сети

В 1952 Вардропом был сформулирован принцип, описывающий ситуацию конкурентного равновесия [29]. Согласно этому принципу, потоки в сети распределяются так, что во время движения по всем используемым маршрутам одинаково и меньше времени свободного движения по неиспользуемым маршрутам между любой фиксированной парой исток-сток.

Конкурентное равновесие – это такое распределение потоков F^w по имеющимся маршрутам R^w, что время передвижения одинаково по всем используемым маршрутам и меньше времени свободного передвижения по неиспользуемым маршрутам из R^w, для каждой пары районов отправления-прибытия $w \in W$ (3.1):

$$\sum_{e \in E} t_e(x_e) * \delta_{e,r}^w \begin{cases} = t^w, \text{если } f_r^w > 0, \\ \geq t^w, \text{если } f_r^w = 0, \end{cases} \forall r \in R^w, \quad (3.1)$$

где t^w– время передвижения по всем используемым маршрутам между парой районов отправления-прибытия $w \in W$ в ситуации, когда потоки распределены согласно первому принципу Вардропа

Впервые задача поиска конкурентного равновесия в транспортной сети была формализована в [29]. Ниже приведена математическая интерпретация процесса поиска конкурентного равновесия.

Предлагается рассмотреть сеть городских дорог, представленную в виде ориентированного графа $G = (E, V)$, где V – множество перекрестков, E – множество дорог между соседними перекрестками [31]. Определим $W \subseteq$

$V \times V$ как упорядоченное множество пар узлов с ненулевым спросом на поездки $F^w > 0$, $w \in W$. W обычно называют множеством пар отправления-прибытия ($OD-$пар), $|W| = m$. Любое множество последовательно связанных дуг, начинающихся в узле-источнике $OD-$пары w и заканчивающихся в конечном узле $OD-$пары w, назовем маршрутом между $OD-$парой $w, w \in W$. Упорядоченное множество всех возможных маршрутов между узлами $OD-$пары w обозначим через $R^w, w \in W$, а упорядоченное множество всех возможных маршрутов между всеми $OD-$парами обозначим через $R, R = \cup_{w \in W} R^w$. Спрос $F^w > 0$ удовлетворяется по доступным маршрутам R^w: $\sum_{r \in R^w} f_r^w = F^w$, где f_r^w – переменная, соответствующая транспортным потокам по маршруту $r \in R^w$ между узлами $OD-$пары $w \in W$. Введем вектор спроса $F = (F^1, \cdots, F^m)^T$ и вектор f, который представляет собой структуру направлений транспортных потоков, таких что $f = f_R = (\cdots, f_r^w, \cdots)^T$ обычно вектор $\{f_r^w\}_{r \in R^w}^{w \in W}$.

Введем дифференцируемые строго возрастающие функции на множестве вещественных неотрицательных чисел $t_e(\cdot), e \in E$. Предположим, что $t_e(\cdot), e \in E$ неотрицательны и их первые производные строго положительны на множестве вещественных неотрицательных чисел. Через x_e обозначим транспортный поток на дуге e, а x соответствующий вектор дуг-потоков, $x = (\cdots, x_e, \cdots)^T, e \in E$. Определенные функции $t_e(x_e)$ используются для описания времени прохождения по дугам $e, e \in E$ и их как правило называют функциями задержки дуги, стоимости или производительности. Мы предполагаем, что функция времени в пути маршрута $r \in R^w$ между $OD-$парой $w, w \in W$ – это сумма задержек в пути на всех дугах, принадлежащих этому маршруту. Таким образом, определим время в пути по маршруту $r \in R^w$ между $OD-$парой $w, w \in W$, как следующую сепарабельную функцию:

$$t_r^w(f) = \sum_{e \in E} t_e(x_e) \delta_{e,r}^w \; \forall r \in R^w, w \in W, \qquad (3.2)$$

где по определению

$$\delta_{e,r}^{w} = \begin{cases} 1, \text{если дуга } e \text{ принадлежит маршруту } r \in R^{w} \\ 0, \text{иначе} \end{cases}, \ \forall\, e \in E, w \in W,$$

при этом естественно

$$x_e = \sum_{w \in W} \sum_{r \in R^w} f_r^w \, \delta_{e,r}^w, \forall\, e \in E,$$

то есть транспортный поток на дуге – это сумма транспортных потоков по всем маршрутам, которые можно включить в эту дугу.

Задача о распределении равновесных транспортных потоков при сепарабельных функциях времени пути имеют следующую математическую постановку:

$$\chi(F) = \min_x \sum_{e \in E} \int_0^{x_e} t_e(u) du, \tag{3.4}$$

при условии

$$\sum_{r \in R^w} f_r^w = F^w, \forall\, w \in W, \tag{3.5}$$

$$f_r^w \geq 0, \forall\, r \in R^w, w \in W, \tag{3.6}$$

где по определению

$$x_e = \sum_{w \in W} \sum_{r \in R^w} f_r^w \, \delta_{e,r}^w, \forall\, e \in E. \tag{3.7}$$

При этом, если в результате решения (3.4) – (3.7) находим *x,* то речь идет о задаче распределения трафика дугового потока [31]. В противном случае, если

выполняется поиск f, то рассматривается задача распределения трафика в равновесном маршруте-потоке [32].

Задача оптимизации пропускной способности транспортной сети, представленной в виде графа G, может быть сформулирована в виде следующей задачи двухуровневой оптимизации:

$$T(x^*, c^*) = \min_{c} \sum_{e \in E} \int_0^{x_e} t_e(x_e, c_e) x_e, \tag{3.8}$$

при ограничениях

$$\sum_{e \in E} c_e \leq C, \tag{3.9}$$

$$c_e \geq \boldsymbol{c}_e, \forall e \in E, \tag{3.10}$$

и условии, что транспортные потоки распределяются согласно первому принципу Вардропа:

$$x^* = \operatorname*{argmin}_{x} \sum_{e \in E} \int_0^{x_e} t_e(u, c_e) du, \tag{3.11}$$

при выполнении условий (3.5) – (3.7).

Исходя из ограничения (3.10) пропускная способность дуги может быть увеличена при условии, что изначальная пропускная способность дуги равна $\boldsymbol{c}_e, e \in E$. Ограничение (3.9) означает, что администрация может увеличить пропускную способность улично-дорожной сети до заданного предела C. Ограничение вызвано возможностями администрации в инвестировании реконструкции. При этом администрация может управлять только пропускными способностями дорог, а значит минимизация на верхнем уровне возможна только по c. Условия (3.5) – (3.7) описывают реакцию участников движения на инфраструктурные изменения.

Решить такую задачу точными методами практически невозможно ввиду того, что она NP-сложная. Также оптимизационная задача невыпуклая, причем если выпуклость задач верхнего и нижнего уровня не гарантируют выпуклость задачи двухуровневой оптимизации.

Существует альтернативная формулировка задачи оптимизации топологии транспортной сети:

$$T(x^*, c^*) = \min_x \sum_{e \in E} t_e(x_e, c_e)\, x_e, \tag{3.12}$$

при ограничениях

$$\sum_{e \in E} c_e \leq C, \tag{3.13}$$

$$c_e \geq \boldsymbol{c}_e, \forall e \in E, \tag{3.14}$$

и условии, что транспортные потоки распределяются согласно первому принципу Вардропа:

$$\min_x \sum_{e \in E} \int_0^{x_e} t_e(u)du, \tag{3.15}$$

при ограничениях

$$\sum_{e_1 \in V_p} x_{e_1}^w - \sum_{e_2 \in W_p} x_{e_2}^w = d_p, \forall p \in V, \tag{3.16}$$

$$x_e^w \geq 0, \forall\, e \in E, w \in W, \tag{3.17}$$

при

$$x_e = \sum_{w \in W} x_e^w, \forall e \in E, \tag{3.18}$$

где V_p – множество дуг, входящих в узел p, а W_p – множество дуг, выходящих из узла p, а

$$d_p^w = \begin{cases} -F^w, & \text{если узел } p \text{ является истоком в паре } w \in W, \\ F^w, & \text{если узел } p \text{ является стоком в паре } w \in W, \\ 0, & \text{иначе.} \end{cases} \quad (3.19)$$

В качестве линейной функции задержки используется BPR-функция в виде (рисунок 26):

$$t_i(f_i) = t_i^0\left(1 + \frac{f_i}{c_i}\right), \forall i = \overline{1,n}.$$

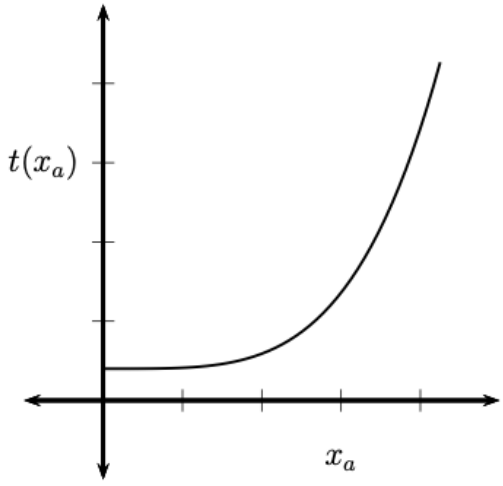


Рисунок 26 – Вид BPR- функции

Здесь c_i выступает в качестве пропускной способности маршрута i, $i = \overline{1,n}$, а явный вид решения задачи распределения потоков сети, соответствующий такой функции задержки, удобнее подставлять в целевой функционал верхнего уровня задачи оптимизации топологии сети.

Таким образом, была рассмотрена математическая постановка двухуровневой задачи оптимизации топологии сети с задачей равновесного распределения транспортных потоков на нижнем уровне. Решить двухуровневую задачу оптимизации топологии транспортной сети, используя точные методы практически невозможно, так как она является NP-сложной.

Было решено подобрать нужную адаптацию для проведения эффективной оптимизации.

3.2. Исследование способов адаптации алгоритма роя частиц для эффективного решения задач оптимизации

Сначала было проанализировано влияние изменения численности роя на эффективность работы и время работы алгоритма. Проверка осуществлялась на следующих тестовых функциях: функции Растригина, Экли, сферы, Бута и Била. Эксперимент проводился для роя, состоящего из 10, 100, 500 и 1000 особей. В таблицах 2 и 3 и на рисунках 26, 27 зафиксированы лучшее минимальное значение, найденное роем и среднее минимальное значение, полученное при реализации алгоритма 100 раз.

Таблица 2. Минимальное значение функции, найденное роем

	10 особей	100 особей	500 особей	1 000 особей
Функция Растригина	1,274	0,833	0,006	0,028
Функция Экли	0,019	0,005	0,0001	0,0004
Функция сферы	0,325	0,0298	0,001	0,003
Функция Бута	$3{,}78*10^{-4}$	$5{,}97*10^{-6}$	$1{,}94*10^{-7}$	$2{,}43*10^{-6}$
Функция Била	$1{,}55*10^{-4}$	$1{,}11*10^{-5}$	$3{,}27*10^{-7}$	$1{,}59*10^{-7}$

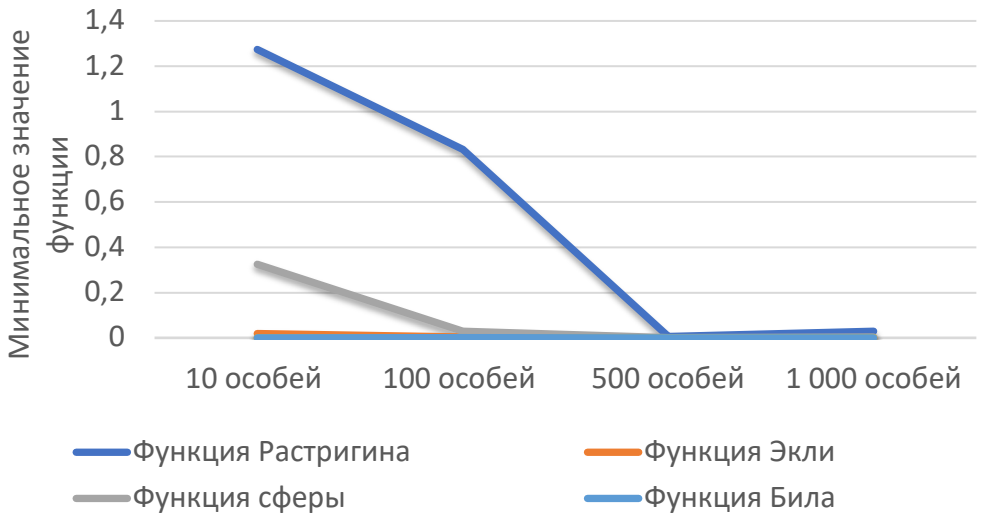


Рисунок 26 – Минимальное значение функции, найденное роем из 10, 100, 500 и 1000 особей

Таблица 3. Среднее минимальное значение функции, найденное роем за 100 повторений алгоритма

	10 особей	100 особей	500 особей	1 000 особей
Функция Растригина	4,450	1,684	0,355	0,164
Функция Экли	0,313	0,052	0,016	0,009
Функция сферы	250,84	11,16	1,78	0,63
Функция Бута	$8{,}7*10^{-2}$	$3{,}0*10^{-3}$	$4{,}2*10^{-4}$	$1{,}9*10^{-4}$
Функция Била	$9{,}3*10^{-2}$	$3{,}2*10^{-4}$	$5{,}1*10^{-5}$	$2{,}1*10^{-5}$

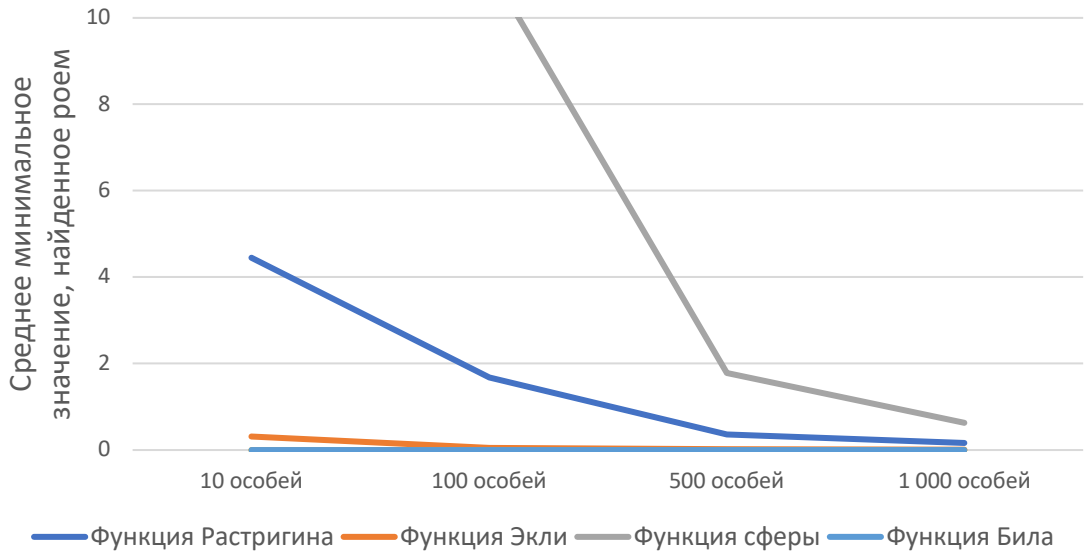


Рисунок 27 – Изменение среднего минимального значения функции, найденного роем, в зависимости от количества особей

Результаты показали, что увеличение частиц в рое приводит к улучшению искомого значения целевой функции. Так, для функции Растригина лучшее найденное значение уменьшилось с 1,274 до 0,028 для 10 и 1000 частиц в рое соответственно. Это позволяет утверждать, что с увеличением частиц в рое увеличивается вероятность, что эксперимент дал результат достаточно близкий к настоящему глобальному экстремуму. Наименьшее уменьшение значения целевой функции наблюдается у функций Бута и Била, следовательно, для них значительное увеличение роя не является целесообразным.

Однако увеличение частиц в рое ведет к увеличению работы алгоритма. В таблице 4 представлено время работы алгоритма при выполнении 100 итераций для роя, состоящего из 10, 100, 500 и 1000 особей.

Таблица 4. Время работы 100 запусков алгоритма

	10	100	500	1 000
Функция Растригина	1,31	9,04	45,17	89,41
Функция Экли	4,84	43,57	205,49	443,06
Функция сферы	4,17	35,59	173,94	362,17
Функция Бута	3,85	31,93	175,85	374,66
Функция Била	4,02	39,25	189,37	390,59

Эффективность увеличения численности роя проверялась по прологарифмированному приращению среднего минимального значения и времени работы алгоритма. Как видно из рисунка 28, в среднем лучшее найденное значение функции уменьшалось быстрее, чем увеличивалось время работы, следовательно увеличение численности роя эффективно. Однако в частном случае для функции Била увеличение численности роя с 500 особей до 1000 уменьшает лучшее найденное значение только на 0,00000017. Выбор параметра численности роя будет варьироваться в зависимости от особенностей каждой конкретной задачи: увеличения времени работы алгоритма, необходимой точности решения.

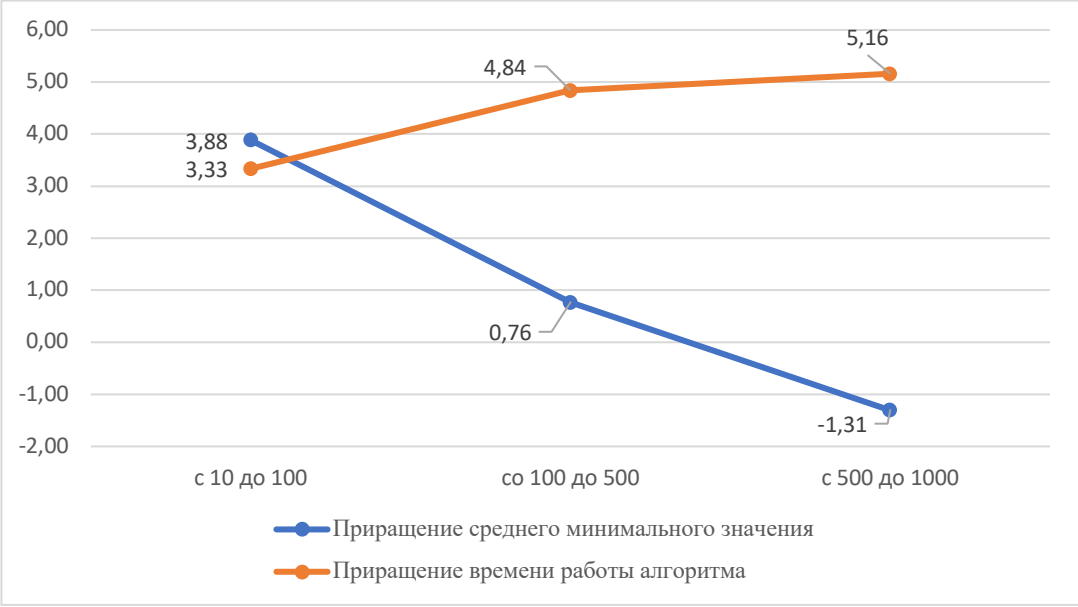


Рисунок 28 – Изменение среднего минимального значения, найденного роем и времени работы алгоритма с увеличением популяции

Также был проанализирован средний, минимальный и максимальный шаг, на котором было получено лучшее решение роем. Результаты представлены в

таблице 5. Лучшее найденное значение на каждой итерации представлено на рисунке 29. В среднем минимальное лучшее значение за 100 итераций вычисляется роем на 40 шаге, однако это значение может варьироваться от 1 шага, когда лучшее найденное значение не менялось с первой итерации, до 98. Увеличение количества итераций также может улучшить результаты работы алгоритма.

Таблица 5. Время работы 100 запусков алгоритма

	10 особей	100 особей	500 особей	1 000 особей
Функция Растригина	43	43	47	42
Функция Экли	39	36	39	38
Функция сферы	36	37	38	42
Функция Бута	41	41	40	40
Функция Била	52	47	43	46

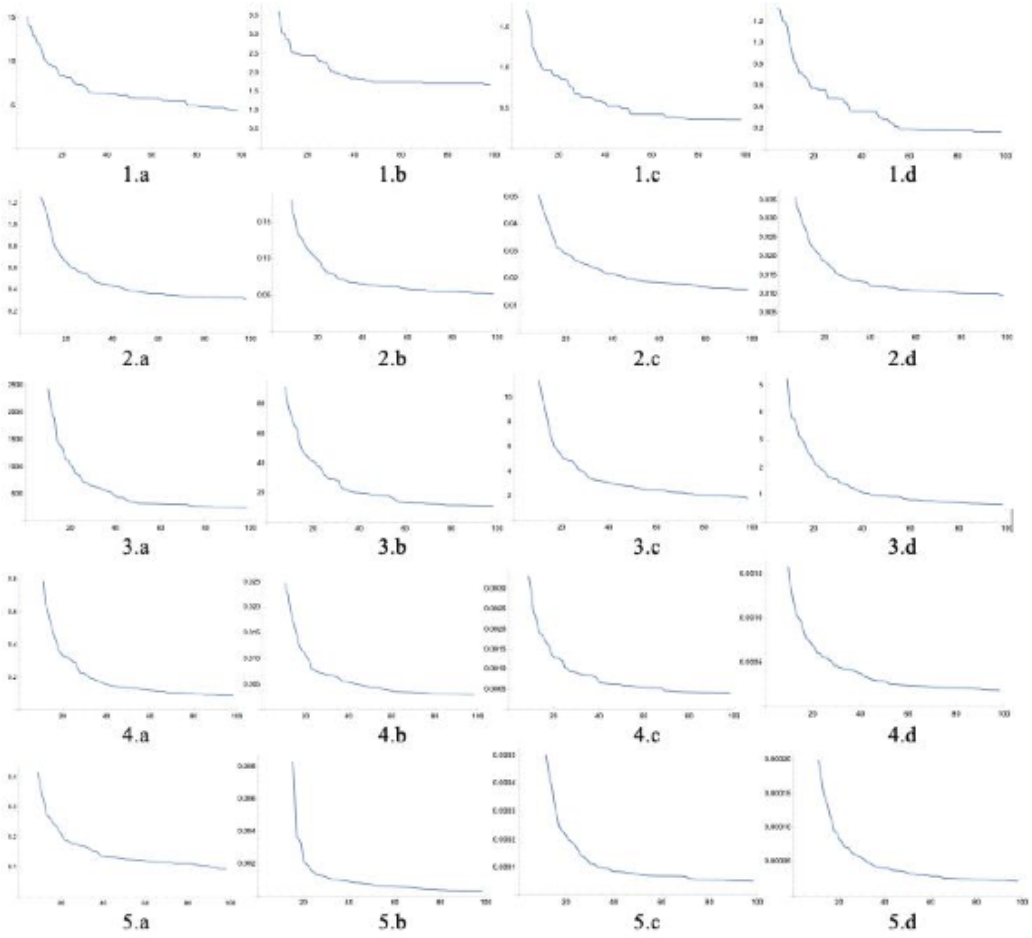


Рисунок 29 – Лучшее решение, найденного роем на каждой итерации для исследуемых функций

Для роя, состоящего из 1000 особей лучшее найденное значение отличалось от настоящего глобального минимума от $1{,}59 \cdot 10^{-7}$ до 0,28 для исследуемых функций. Это позволяет говорить об эффективности алгоритма в нахождении глобального минимума функций разного вида, в том числе с большим количеством локальных экстремумов. При этом такие параметры, как количество особей в рое и количество итераций, следует варьировать для каждой конкретной решаемой задачи.

После этого был проведен изменения коэффициента инерции. Лучший результат среди линейных показала модификация коэффициента инерции по формуле (1.4). На рисунке 13.а оказано, как изменялись лучшие значения, найденные роем за 100 итераций на примере 9 вызовов алгоритма.

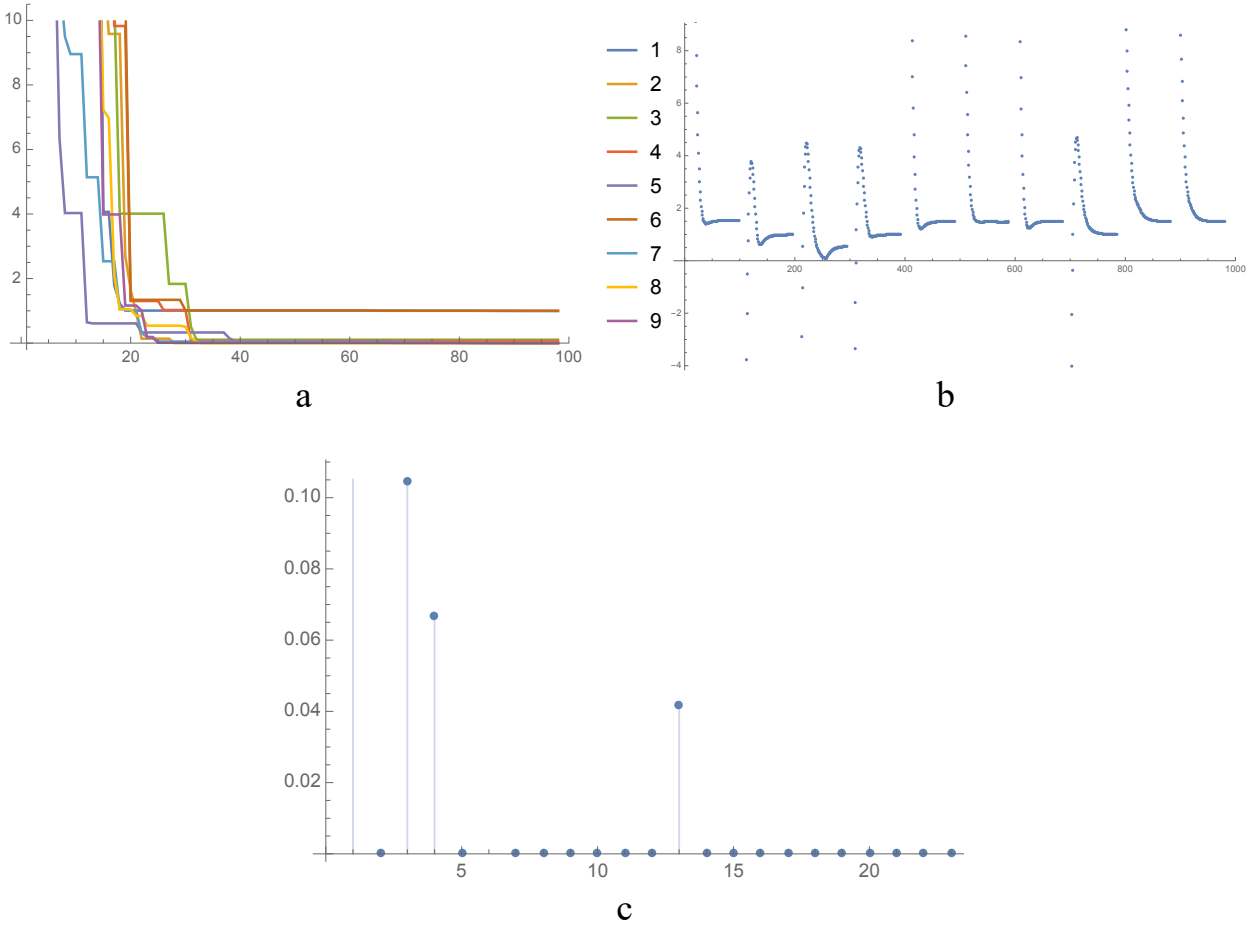


Рисунок 30 – Модификация с использованием линейного коэффициента инерции вида 1.4

Видно, что уже к 40 итерации находилось минимальное значение, однако в случае 6 увеличение количества итераций могло привести к нахождению меньшего значения функции. Рисунок 13.b отражает траектории изменения лучшего значения, найденного каждой частей в рое. Алгоритм вызывался 30 раз, на рисунке 13.c показано лучшее значение, найденное в каждом применении. Оно варьируется от $4{,}30234 * 10^{12}$ до 1,00186. Среднее минимальное значение – 0,09611.

Для этой модификации был проведен анализ размера роя. Количество частиц в рое было равным – 3, 5, 7, 10, 15, 20, 30, 40, 50. На рисунке 31.a показано изменение лучшего найденного значения алгоритмом. Оно также перестает изменяться к 40 итерации, однако в случае, если в рое 3 частицы, лучшее найденное значение намного больше минимального. Алгоритм вызывался 9 раз, на рисунке 14.c показано лучшее значение, найденное в каждом применении. Худший результат показал алгоритм с 3 частицами в рое – 1,45125, лучший – $44.99705 * 10^{-9}$. Как показал анализ, с того момента, когда в рое было 20 и более частиц их увеличение не влияло на качество работы алгоритма.

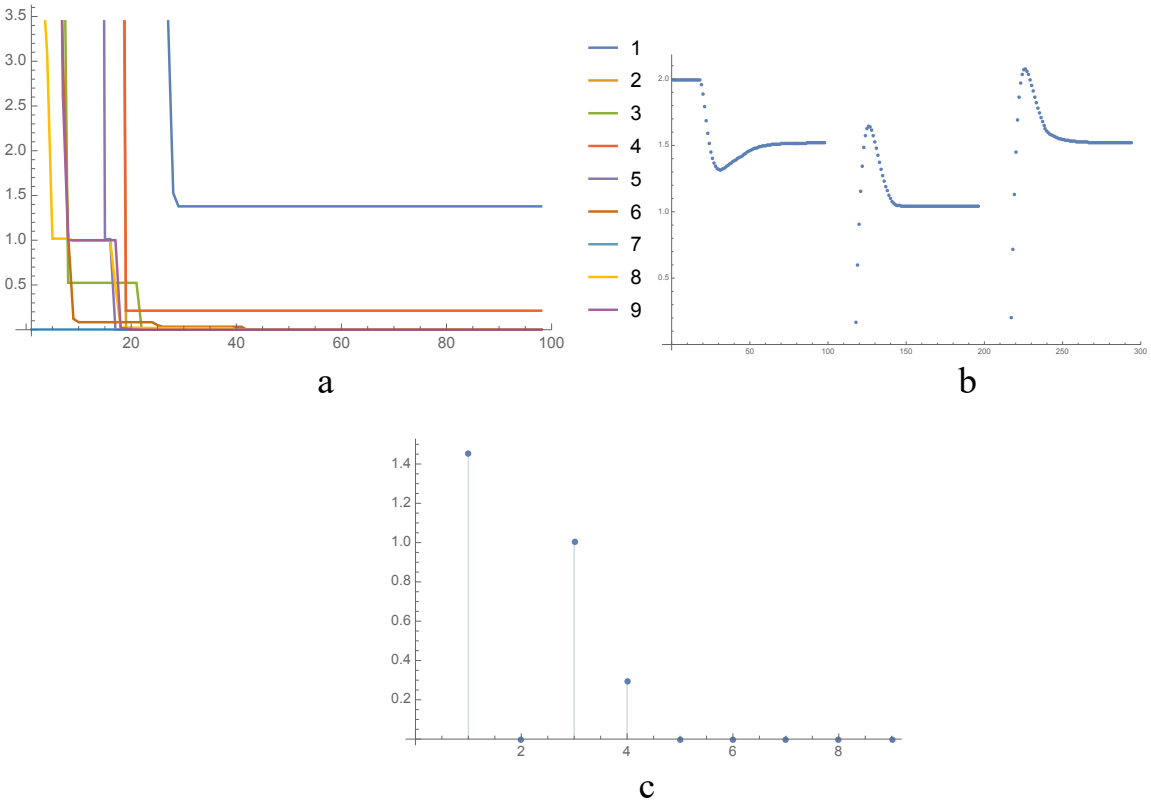
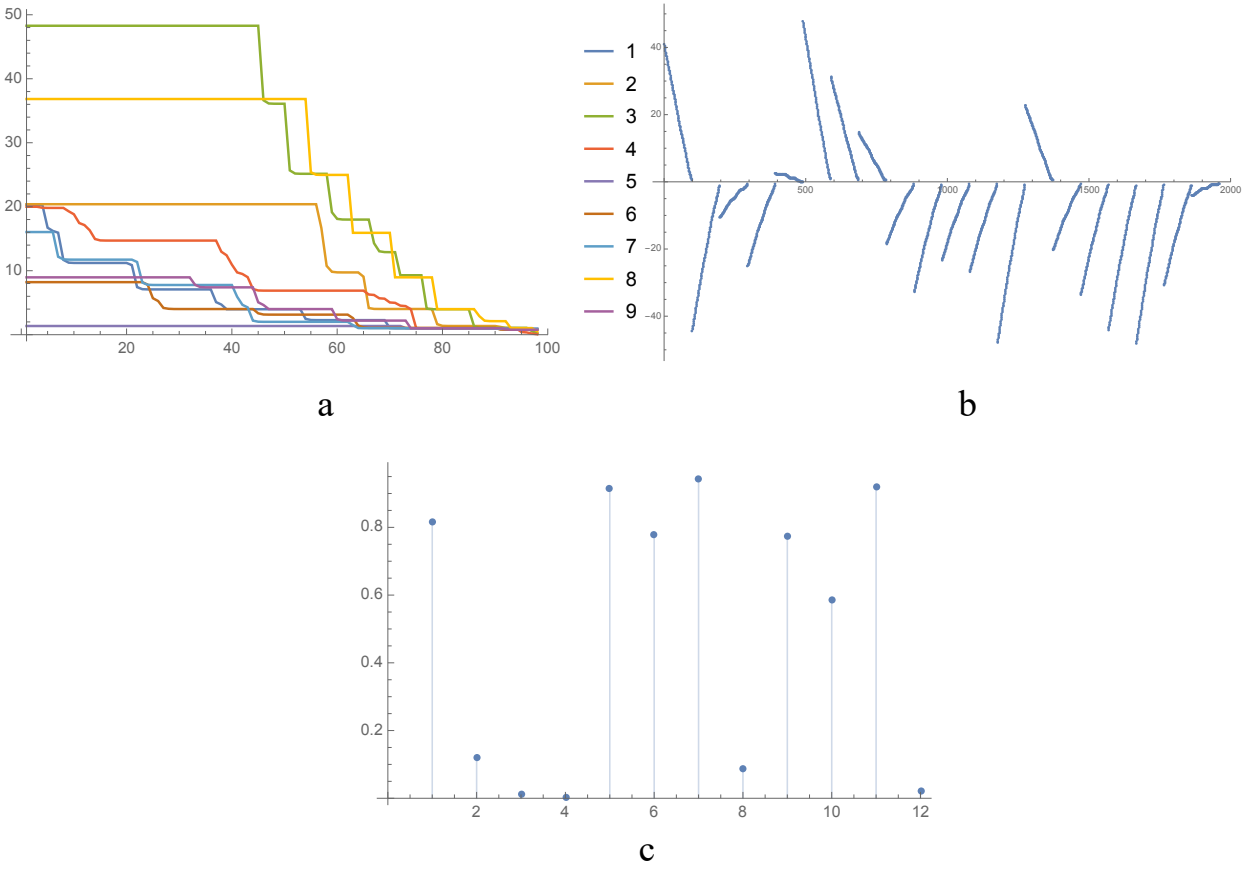


Рисунок 31 – Влияние изменения частиц в рое

Стоит отметить, что нелинейное изменение коэффициента инерции не увеличило эффективность работы алгоритма. На рисунках 32.a и 32.b показано изменение лучшего найденного значения функции на каждой итерации и изменение лучшего значения, найденного каждой из частиц соответственно. На рисунке 32.c показаны результаты нескольких применений алгоритма для нахождения минимума функции. Среднее минимальное значение функции, найденное за все вызовы алгоритма равно 0,800098.



a b

c

Рисунок 32 - Модификация с использованием нелинейного коэффициента инерции

Таким образом, в результате проведения исследования адаптаций алгоритма было решено использовать динамический линейный коэффициент инерции вида (1.4); численность роя принять равной 20; использовать полносвязную структуру топологию соседства частиц на этапе исследования пространства решения и кольцевую на этапе эксплуатации.

3.3. Описание алгоритма PSODP – LeBlanc с динамическим коэффициентом инерции

Подходы к оптимизации верхнего и нижнего уровней рассмотренной задачи отличаются. На верхнем уровне предлагается нахождение оптимального значения функции с помощью роевого интеллекта с подобранными адаптациями, на нижнем уровне, оптимизация реализуется алгоритмом LeBlanc. Общая схема работы алгоритма PSODP–LeBlanc с динамическим коэффициентом инерции вынесена в приложение А.

В данном случае каждая частица является одним их возможных вариантов переменной с – матрицы пропускной способности сети. На первом этапе происходит инициализация переменных верхнего уровня, где ограничение – это нижняя и верхняя границы. Нижняя граница – это пропускная способность при нулевой загрузке сети. Верхняя представляет собой вариант, когда дальнейшее увеличение загрузки сети ведет к экспоненциальному росту времени движения по ней участников. Затем алгоритм LeBlanc вычисляет переменные нижнего уровня, а именно равновесное распределение транспортных потоков на сети. После происходит пересчет значений целевых функций верхнего и нижнего уровней, обновление лучших позиций у каждой отдельной частицы, а также лучшего значения целевой функции верхнего уровня и лучшего положения переменных верхнего уровня для всего роя.

Оптимизация задачи верхнего уровня.

Особенностями оптимизации верхнего уровня являются подобранные адаптации алгоритма к задаче:

1. Численность роя. Она составила 20 особей для того, чтобы обеспечить достаточное разнообразие и при этом сильно не увеличивать время работы алгоритма. На оптимизацию функции нижнего уровня для каждой частицы тратятся значительные вычислительные мощности, что вызывает необходимость сокращать популяцию на сколько это возможно.

2. Топология соседства частиц. Алгоритм на верхнем уровне – это динамический алгоритм роя частиц. На ранних этапах работы он использует

полносвязную структуру соседства для более быстрого перемещения частиц по пространству поиска. После того, как было пройдено 65% итераций алгоритм начинает использовать кольцевую топологию, что уменьшает скорость перемещения частиц. Связанными остаются частицы с наиболее близкими текущими значениями целевой функции верхнего уровня. На рисунке 33 изображен переход от полносвязной структуры соседства к кольцевой.

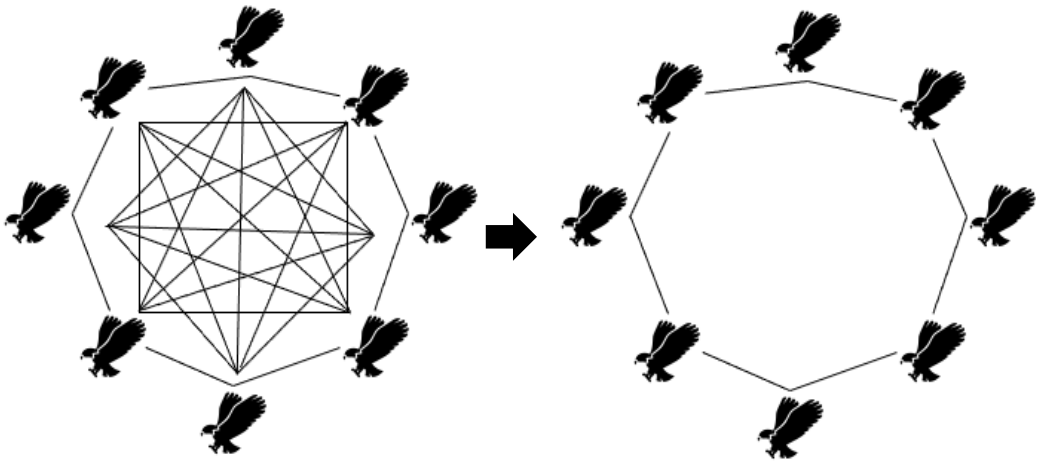


Рисунок 33 – Иллюстрация перехода от кольцевой к полносвязной топологии соседства частиц в динамической фазе

3. Динамический коэффициент инерции. В качестве коэффициента инерции был выбран линейно убывающий, который определяется по формуле (1.4). Однако в случае, если лучшее значение функции не менялось в течении 40 итераций коэффициент становится линейно возрастающим и увеличивается случайная составляющая в пересчете скорости на несколько итераций, что обеспечивает перемещение частиц из локальных минимумов в другие точки на пространстве поиска.

Обновление положения частиц происходит по формулам (1.1), (1.2).

Алгоритм LeBlanc для оптимизации равновесного распределения транспортных потоков на нижнем уровне [33].

Для того чтобы начать оптимизацию, на вход подается найденная матрица емкости сети.

1. Предположим, что текущее допустимое решение – x^k.

2. Вычисляем $A_{i,j}(x_y)|_{x=x}^{k} \equiv c_{i,j}.\ y_{i,j} = 0\ \forall(\mathrm{i},\mathrm{j}),\ \mathrm{s} = 1.$

3. Находим кратчайший путь между каждым узлом на сети и узлом s, то есть решаем. Задачу о кратчайшем маршруте, используя $c_{i,j}$ как расстояния.

4. Для каждого источника r в сети выполнить операцию $y_{i,j} = y_{i,j} + D(r,s)$ для каждого узла (i,j) на кратчайшем маршруте из узла r в узел s.

5. Если s не равно количеству узлов в маршруте, $s = s + 1$ и возвращаемся к шагу 3, иначе переходим к шагу 6.

6. Теперь $y_{i,j}$ решение подзадачи k. Через y^k обозначим вектор $y_{i,j}$. Минимизируем функцию f на отрезке между x^k и y^k, используя технику одномерного поиска. Полученное решение обозначим через x^{k+1}.

7. Проверяем критерий остановки алгоритма и переходим к шагу 2.

После того, как мы получили матрицу загрузки сети x, происходит пересчет значений целевых функций верхнего и нижнего уровней, а так же обновление лучших значений. Далее проверяется условие прекращения работы алгоритма. В случае если оно выполняется, за решение двухуровневой задачи принимаем положение частиц, в котором получено текущее лучшее значение целевой функции верхнего уровня, и соответствующую ему матрицу x. Иначе возвращаемся к оптимизации целевой функции верхнего уровня.

Таким образом, в параграфе был рассмотрен алгоритм, который был применен для решения двухуровневой задачи оптимизации топологии транспортной сети. Он сочетает в себе роевые эвристики и адаптации на верхнем уровне, задача нижнего уровня решается алгоритмом LeBlanc.

3.4. Реализация алгоритма PSO PSODP – leBlanc с динамическим коэффициентом инерции и его применении на сети Sioux-Falls

Задача двухуровневой оптимизации топологии транспортной сети выглядит следующим образом. Целевая функция (3.20) затрат на перемещение по сети состоит из двух частей, где первая – это время затрачиваемой на

преодоление пути в виде BPR-функции, а вторая – это функция затрат, сумма по которым не должна превышать 500 млн.

$$\min_{c} \sum_{i=1}^{n} t_i(x_i, c_i)x_i + 0{,}01 \sum_{i=1}^{n} g_i(c_i) \tag{3.20}$$

Заданы нижняя и верхняя границы переменной c_i – доступной емкости сети. 360600

$$\min_{x} \sum_{i=1}^{n} \int_0^{x_i} t_i(u, c_i)du \tag{3.21}$$

Для проверки алгоритма была выбрана сеть Sioux-Falls [34]. Ее можно представить в виде ориентированного взвешенного графа, состоящего из 24 узлов и 74 дуг. Визуализация тестовой сети представлена на рисунке 34.

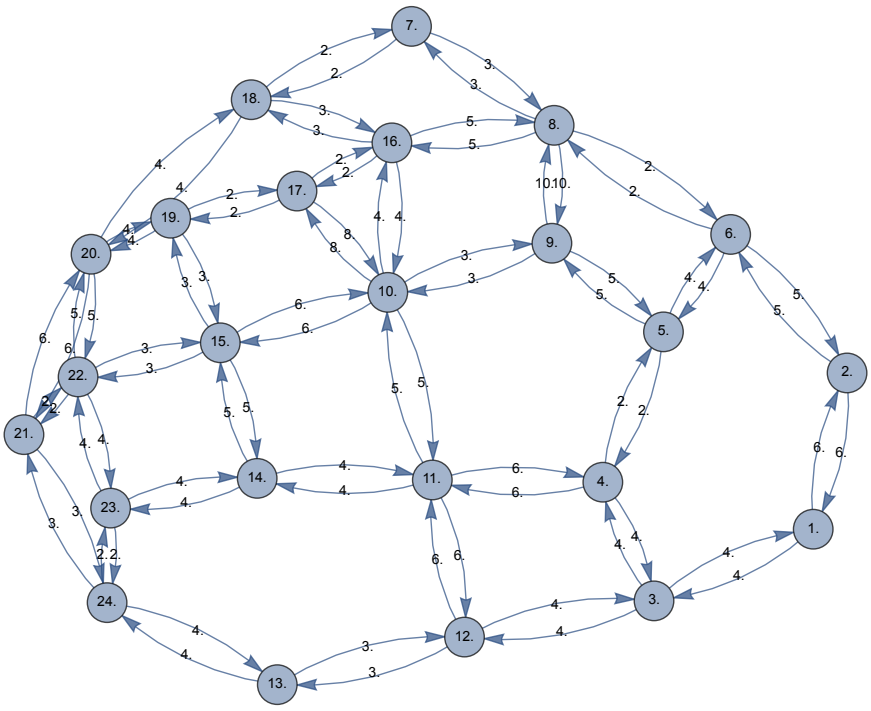


Рисунок 34 – Визуализация сети Sioux-Falls

Исходные данные, содержащие информацию о сети, вынесены в приложение Б. На вход подаются значения верхней и нижней границы пропускной способности, матрица спроса на перемещение между узлами. Все три матрицы размерности имеют размерность 24×24 по количеству узлов в сети. Визуализация размера спроса на перемещение между узлами сети представлена на рисунке 35.

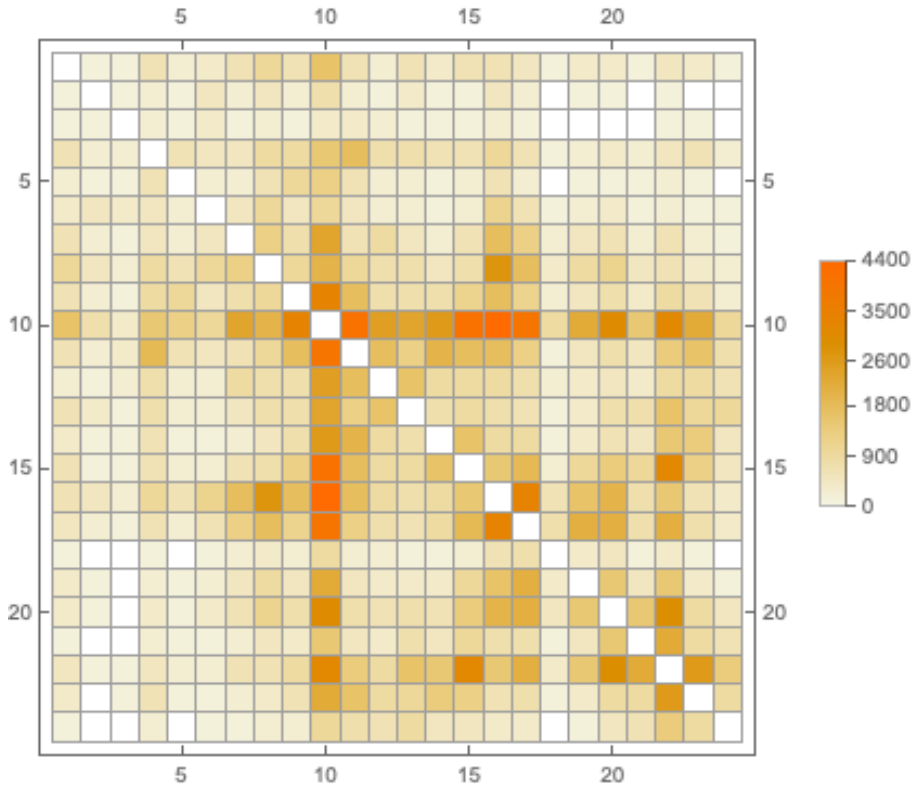


Рисунок 35 – Визуализация размеров спроса на перемещение между узлами

В алгоритме значения коэффициентов $wmin, wmax, c_1$ и c_2 равны 0,3, 1, 0,2 и 0,2 соответственно, r_1 и r_2 на каждой итерации принимают случайное значение на интервале (0;1), количество особей в рое – 20. Всего было реализовано 130 итераций.

Полученная в результате пропускная способность дуг и их равновесная загрузка вынесены в приложение В и приложение Г соответственно. Динамика уменьшения лучшего найденного значения целевой функции верхнего уровня представлена на рисунке 36. Путем оптимизации удалось уменьшить целевую функцию с 393,701 до 384,141. Переход алгоритма от фазы исследования к фазе эксплуатации позволил найти более точное локальное решение. На итерациях

110–116 лучшее решение уменьшалось на 0,0000386826, 0,000032401, 0,0000271865, 0,0000229472, 0,0000194826, 0,0000166364 соответственно, после чего изменение прекратилось.

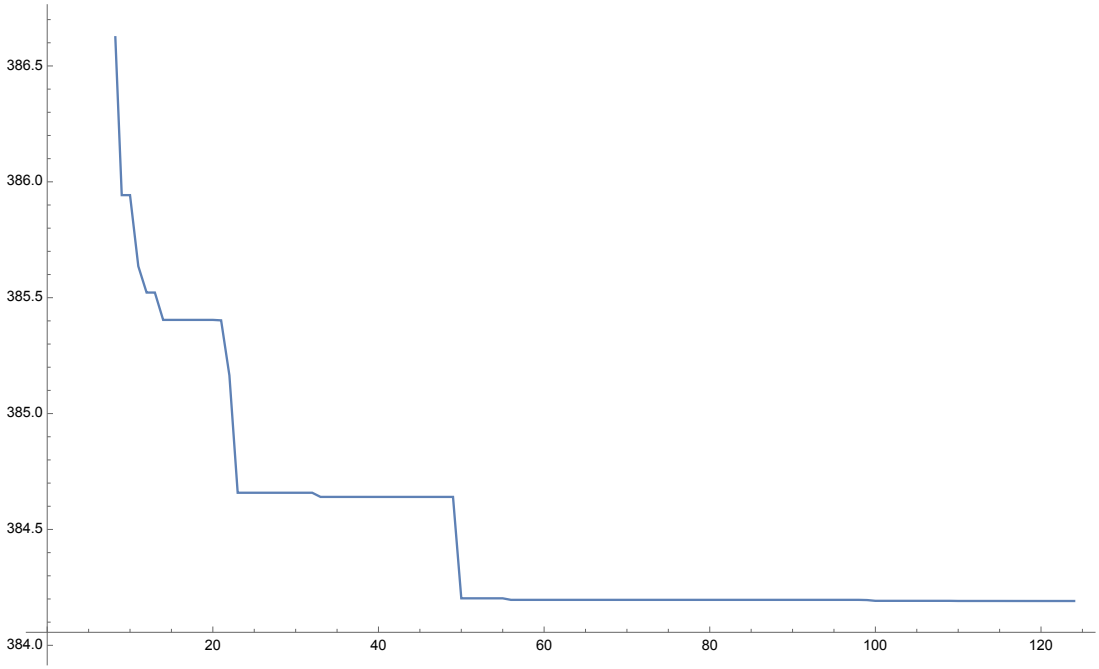


Рисунок 36 – Динамика изменения лучшего найденного значения целевой функции роем

Таким образом, был реализован алгоритм PSODP – LeBlanc для задачи оптимизации топологии сети. Количества особей в рое было достаточно для эффективного исследования пространства поиска. Динамика изменения и непосредственно лучшее решение, найденное каждой отдельной частицей, вынесены в приложение Д. Минусом алгоритма является большая скорость вычисления равновесного распределения транспортных потоков на нижнем уровне. Однако это дает возможность получить более точное решение оптимизируемой функции, что оправдывает затраченные вычислительные мощности.

Заключение

Таким образом, цель работы можно считать достигнутой, а задачи выполненными. В ходе выполнения работы была изучена общая постановка задач двухуровневой оптимизации, а также двухуровневая задача оптимизации топологии сети. Были реализованы алгоритмы BLEAQ и BPSOQ, проведен их сравнительный анализ. Также был реализован алгоритм PSODP–LeBlanc с динамическим коэффициентом инерции, были подобраны адаптации, эффективно решающие исследуемую задачу. Код всех реализованных алгоритмов доступен по ссылке [28].

В первой был проведен обзор особенностей эволюционного отбора и роевого интеллекта из которого видно, что концепции этих подходов отличаются как на уровне природных механизмов, так и математически.

Алгоритмы роевого интеллекта принципиально отличаются от эволюционных, так как не требуют создания новой популяции на каждой итерации через эволюционные механизмы. Они используют коллективные децентрализованные перемещения членов популяции. Был предложен способ отнесения алгоритмов к роевым: в формулах, которые описывают миграцию агентов роя, необходимо наличие объекта, который дает возможность косвенного обмена информацией между ними.

Область применения алгоритмов роевого интеллекта помимо задач оптимизации дополняется управлением группами роботов. Данное направление носит название роевой робототехники и активно развивается в последние годы. Управление роботами на основе эволюционных процессов невозможно, так как для этого необходимо было бы организовать создание новых роботов на основе старых непосредственно во время управления.

Во второй главе были описаны результаты реализации алгоритмов, основанных на роевых и эволюционных подходам и проанализирована эффективность их работы на примере решения задачи двухуровневой оптимизации. В результате проведения сравнительного анализа алгоритмов

BLEAQ и BPSOQ для решения двухуровневых задач оптимизации на тестовых примерах более эффективно показали себя роевые эвристики.

В первую очередь это объясняется что в эволюционном алгоритме потомками, участвующими в соревновании на место в популяции являются две ее случайные особи. В какой-то момент потомки оказываются хуже своих родителей в результате чего обновление популяции прекращается. Кроме того, алгоритм роя частиц реализован на идее коллективного поведения и общение всех участников популяции между собой позволяет более тщательно исследовать пространство поиска, захватывая большие расстояния в начале работы алгоритма и более глубоко исследуя локальные минимумы в конце. Для двухуровневой задачи оптимизации топологии транспортной сети было решено использовать именно роевой интеллект, подобрав перед этим наиболее эффективную адаптацию.

В третьей главе была дана математическая математическая постановка двухуровневой задачи оптимизации топологии сети с задачей равновесного распределения транспортных потоков на нижнем уровне. Были подобраны нужные адаптации для проведения эффективной оптимизации.

В результате проведения исследования адаптаций алгоритма было решено:

- использовать динамический линейный коэффициент инерции вида (1.4);
- численность роя принять равной 20;
- использовать полносвязную структуру топологию соседства частиц на этапе исследования пространства решения и кольцевую на этапе эксплуатации.

Был реализован алгоритм PSODP – LeBlanc для задачи оптимизации топологии сети. Количества особей в рое было достаточно для эффективного исследования пространства поиска. Динамика изменения и непосредственно лучшее решение, найденное каждой отдельной частицей, вынесены в приложение Д. Минусом алгоритма является большая скорость вычисления равновесного распределения транспортных потоков на нижнем уровне. Однако это дает возможность получить более точное решение оптимизируемой функции, что оправдывает затраченные вычислительные мощности.

Список использованных источников

1. Скиена С. Алгоритмы. Руководство по разработке: пер. с англ. / С. Скиена. –2-е изд. – СПб.: БХВ-Петербург, 2013. 720 с.
2. Карпенко А.П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: учебное пособие. Изд-во МГТУ им. Н.Э. Баумана, 2017. 446 с.
3. Матренин П.В. Методы стохастической оптимизации: учебное пособие. Новосибирск. Изд-во НГТУ, 2016 . 67с.
4. Матренин П.В., Секаев В.Г. Системное описание алгоритмов роевого интеллекта // Программная инженерия. 2013. №12. С. 39 – 45.
5. Holland J.H. Adaptation in natural and artificial systems // University of Michigan Press. 1975. 233 P.
6. Андреев А.А. Применение генетических алгоритмов при оптимизации нелинейных функций. Вестник ТГУ, 2009. №5. С. 1036 – 1040.
7. Генетические алгоритмы – математический аппарат [Электронный ресурс] – URL: https://loginom.ru/blog/ga-math (дата обращения: 14.05.2022)
8. Операторы генетического алгоритма [Электронный ресурс] – URL: http://lazysmart.ru/iskusstvenny-j-intellekst/geneticheskie-operatory/ (дата обращения: 14.05.2022)
9. Craig R. Flocks, herds, and schools: a distributed behavioral model // Computer Graphics. 1987. P. 25–34.
10. Kennedy J., Eberhart R. Particle swarm optimization // Proceedings of International Conference of Neural Networks. 1995. P. 1942-1948.
11. Heppner F., Grenander U. A stochastic nonlinear model for coordinated bird flocks // The Ubiquity of Chaos. 1990. P. 233–238.
12. Alaia B., Harbaoui I., Borne P., Bouchriha H. A comparative study of the PSO and GA for the m-MDPDPTW // International Journal of Computers Communications Control. 2018. Vol. 13. No 1. P. 8–23.
13. Shi Y., Eberhart R. A modified particle swarm optimizer // International Conference on Evolutionary Computation Proceedings. 1998. P. 69–73.

14. Shi Y., Eberhart R. Empirical study of particle swarm optimization // Congress on Evolutionary Computation. 1999. P. 1945–1950.

15. Harrison K., Engelbercht A. Interia weight control strategies for particle swarm optimization // Swarm Intelligence. 2019. No 10. P. 267–305.

16. Li Z, Tan G. A self-adaptive mutation-particle swarm optimization algorithm // Proceedings of the Fourth International Conference on Natural Computation. 2009 P. 30–34

17. Chen G., Min Z., Jia J., Xinbo H. Natural exponential inertia weight strategy in particle swarm optimization // Proceedings of the 6th World Congress on Intelligent Control and Automation. 2006. P. 3672–3675.

18. Jiao B., Lian Z., Gu X. A dynamic inertia weight particle swarm optimization algorithm // Chaos, Solitons and Fractals. 2008. P. 698–705.

19. Liu Q, Wei W, Yuan H, Zhan ZH, Li Y Topology selection for particle swarm optimization // Information Sciences. 2015. P. 154–157

20. Lynn N., Ali M., Suganthan P. Population topologies for particle swam optimization and differential evolution// Swarm and Evolutionary Computation 2018. P. 24-35.

21. Lynn N., Suganthan P. Heterogeneous comprehensive learning particle swarm optimization with enhanced exploration and exploitation// Swarm and Evolutionary Computation. 2015. P. 11–24.

22. Munoz-Zavala A., Villa-Diharce E. A new neighborhood topology for the particle swarm optimization algorithm// Particle Swarm Optimization. Theory, techniques and applications - Engineering Tools, Techniques and Tables. 2011. P. 227-247.

23. Li J., Sun Y., Hou S., Particle swarm optimization algorithm with multiple phases for solving continuous optimization problems// Discrete Dynamic in Nature and Society. June 2021. Vol. 13. P. 1-13.

24. Карпенко А.П., Селиверстов Е.Ю. Обзор методов роя частиц для задачи глобальной оптимизации// Машиностроение и компьютерные технологии. 2009. С. 1-26.

25. Sinha A., Malo P., Deb K. A review on bilevel optimization: from classical to evolutionary approachesand applications // Transactions on evolutionary computation. 2018. P. 276–295.

26. Sinha A., Malo P. Efficient evolutionary algorithm for single-objective bilevel optimization // Transaction on evolutionary computation. 2013. P. 276-295.

27. Eiben A.E., Smith, J.E. Introduction to Evolutionary Computing // Berlin Heidelberg: Springer Natural Computing Series. 2015. 287 p.

28. Роевые и эволюционные алгоритмы для решения задач двухуровневой оптимизации – [Электронный ресурс] – URL: https://github.com/daturare/swarm-and-evolutionary-algorithms-for-bilevel-optimization-problem

29. Wardrop J.D. Some theoretical aspects of road traffic research // Proc. Institution of Civil Engineers. 1952. Vol. 2. P. 325–387.

30. Beckmann M.J., McGuire C.B., Winsten C.B. Studies in the economic of transportation // New Haven, CT: Yale University Press. 1956. 359 p.

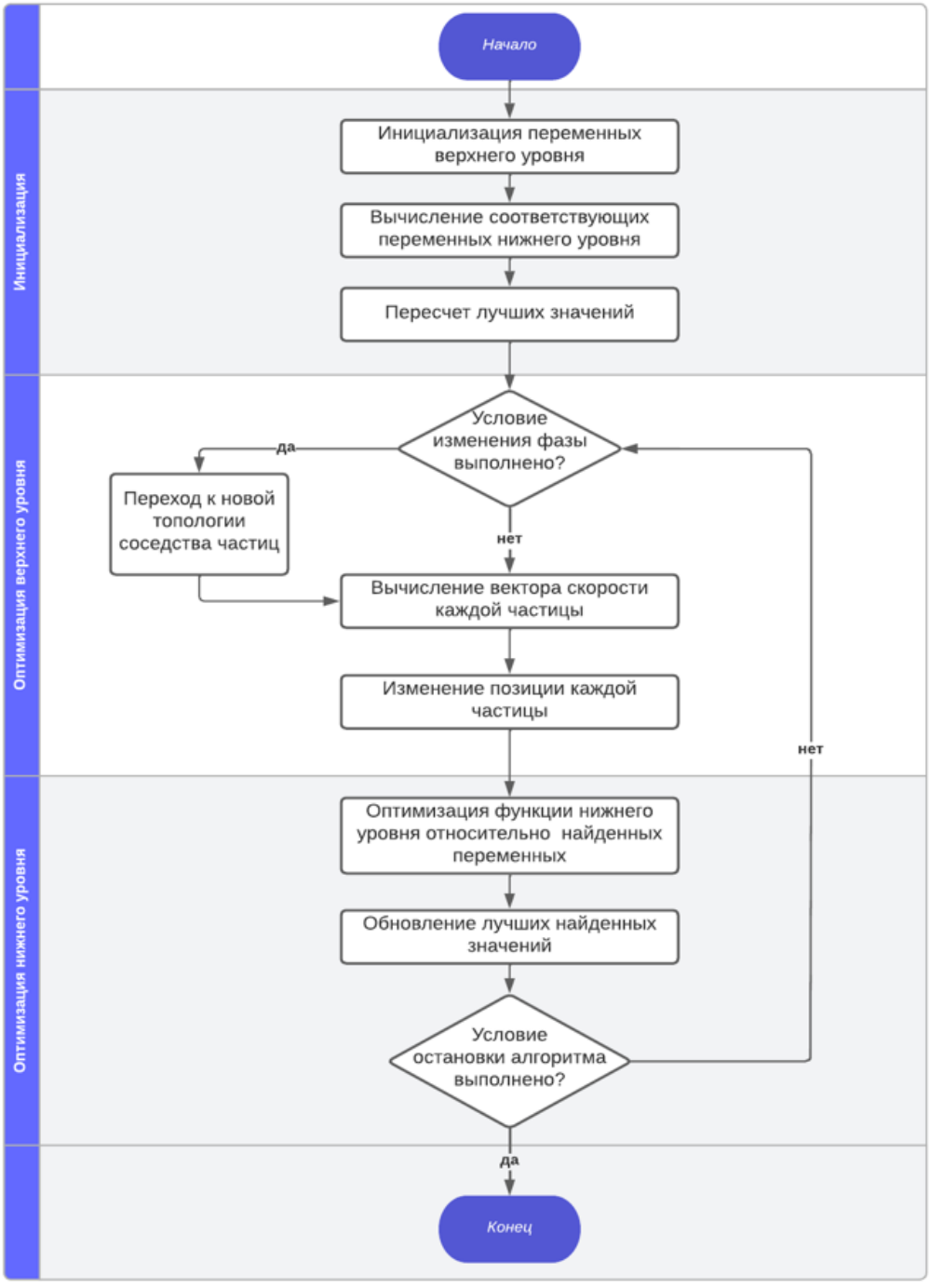
31. Krylatov, A. Reduction of a minimization problem for a convex separable function with linear constraits to a fixed point problem // Journal of Applied and Industrial Mathematics. 2018. P. 98–111.

32. Krylatov, A., Shirokolobova, A. Projection approach versus gradient descent for network's flows assignment problem // Lecture Notes in Computer Science. 2017. P. 345–350 .

33. LeBlanc L.J. An efficient approach to solving the road network equilibrium traffic assignment problem // Transportation research. 1975. P. 309–318.

34. Transportation Networks Sioux Falls – [Электронный ресурс] – URL: https://github.com/bstabler/TransportationNetworks/tree/master/SiouxFalls (дата обращения: 10.05.2022)

Приложение А. Общая схема работы алгоритма PSODP–LeBlanc с динамическим коэффициентом инерции



Приложение Б. Исходные данные для обратной задачи алгоритма

Матрица D - матрица спроса между узлами

0	100	100	500	200	300	500	800	500	1300	500	200	500	300	500	500	400	100	300	300	100	400	300	100
100	0	100	200	100	400	200	400	200	600	200	100	300	100	100	400	200	0	100	100	0	100	0	0
100	100	0	200	100	300	100	200	100	300	300	200	100	100	100	200	100	0	0	0	0	100	100	0
500	200	200	0	500	400	400	700	700	1200	1400	600	600	500	500	800	500	100	200	300	200	400	500	200
200	100	100	500	0	200	200	500	800	1000	500	200	200	100	200	500	200	0	100	100	100	200	100	0
300	400	300	400	200	0	400	800	400	800	400	200	200	100	200	900	500	100	200	300	100	200	100	100
500	200	100	400	200	400	0	1000	600	1900	500	700	400	200	500	1400	1000	200	400	500	200	500	200	100
800	400	200	700	500	800	1000	0	800	1600	800	600	600	400	600	2200	1400	300	700	900	400	500	300	200
500	200	100	700	800	400	600	800	0	2800	1400	600	600	600	900	1400	900	200	400	600	300	700	500	200
1300	600	300	1200	1000	800	1900	1600	2800	0	4000	2000	1900	2100	4000	4400	3900	700	1800	2500	1200	2600	1800	800
500	200	300	1500	500	400	500	800	1400	3900	0	1400	1000	1600	1400	1400	1000	100	400	600	400	1100	1300	600
200	100	200	600	200	200	700	600	600	2000	1400	0	1300	700	700	700	600	200	300	400	300	700	700	500
500	300	100	600	200	200	400	600	600	1900	1000	1300	0	600	700	600	500	100	300	600	600	1300	800	800
300	100	100	500	100	100	200	400	600	2100	1600	700	600	0	1300	700	700	100	300	500	400	1200	1100	400
500	100	100	500	200	200	500	600	1000	4000	1400	700	700	1300	0	1200	1500	200	800	1100	800	2600	1000	400
500	400	200	800	500	900	1400	2200	1400	4400	1400	700	600	700	1200	0	2800	500	1300	1600	600	1200	500	300
400	200	100	500	200	500	1000	1400	900	3900	1000	600	500	700	1500	2800	0	600	1700	1700	600	1700	600	300
100	0	0	100	0	100	200	300	200	700	200	200	100	100	200	500	600	0	300	400	100	300	100	0
300	100	0	200	100	200	400	700	400	1800	400	300	300	300	800	1300	1700	300	0	1200	400	1200	300	100

300	100	0	300	100	300	500	900	600	2500	600	500	600	500	1100	1600	1700	400	1200	0	1200	2400	700	400
100	0	0	200	100	100	200	400	300	1200	400	300	600	400	800	600	600	100	400	1200	0	1800	700	500
400	100	100	400	200	200	500	500	700	2600	1100	700	1300	1200	2600	1200	1700	300	1200	2400	1800	0	2100	1100
300	0	100	500	100	100	200	300	500	1800	1300	700	800	1100	1000	500	600	100	300	700	700	2100	0	700
100	0	0	200	0	100	100	200	200	800	600	500	700	400	400	300	300	0	100	400	500	1100	700	0

Матрица T0 – матрица времени движения участника потока между узлами

0	6	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	4	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0
0	0	4	0	2	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	2	0	4	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	5	0	0	4	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
0	0	0	0	0	2	3	0	10	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0
0	0	0	0	5	0	0	10	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	3	0	5	0	0	0	6	4	8	0	0	0	0	0	0	0
0	0	0	6	0	0	0	0	0	5	0	6	0	4	0	0	0	0	0	0	0	0	0	0
0	0	4	0	0	0	0	0	0	0	6	0	3	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	4
0	0	0	0	0	0	0	0	0	0	4	0	0	0	5	0	0	0	0	0	0	0	4	0
0	0	0	0	0	0	0	0	0	6	0	0	0	5	0	0	0	0	3	0	0	3	0	0

0	0	0	0	0	0	0	5	0	4	0	0	0	0	0	0	2	3	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	2	0	0	2	0	0	0	0	0
0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	3	0	0	0	4	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	2	0	0	4	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	0	6	5	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	2	0	3
0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	5	2	0	4	0	
0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	4	0	2
0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	3	0	2	0

Матрица C – матрица предельных значений суммарного потока между узлами, превысив который, время движения участников потока между узлами начинает сильно возрастать

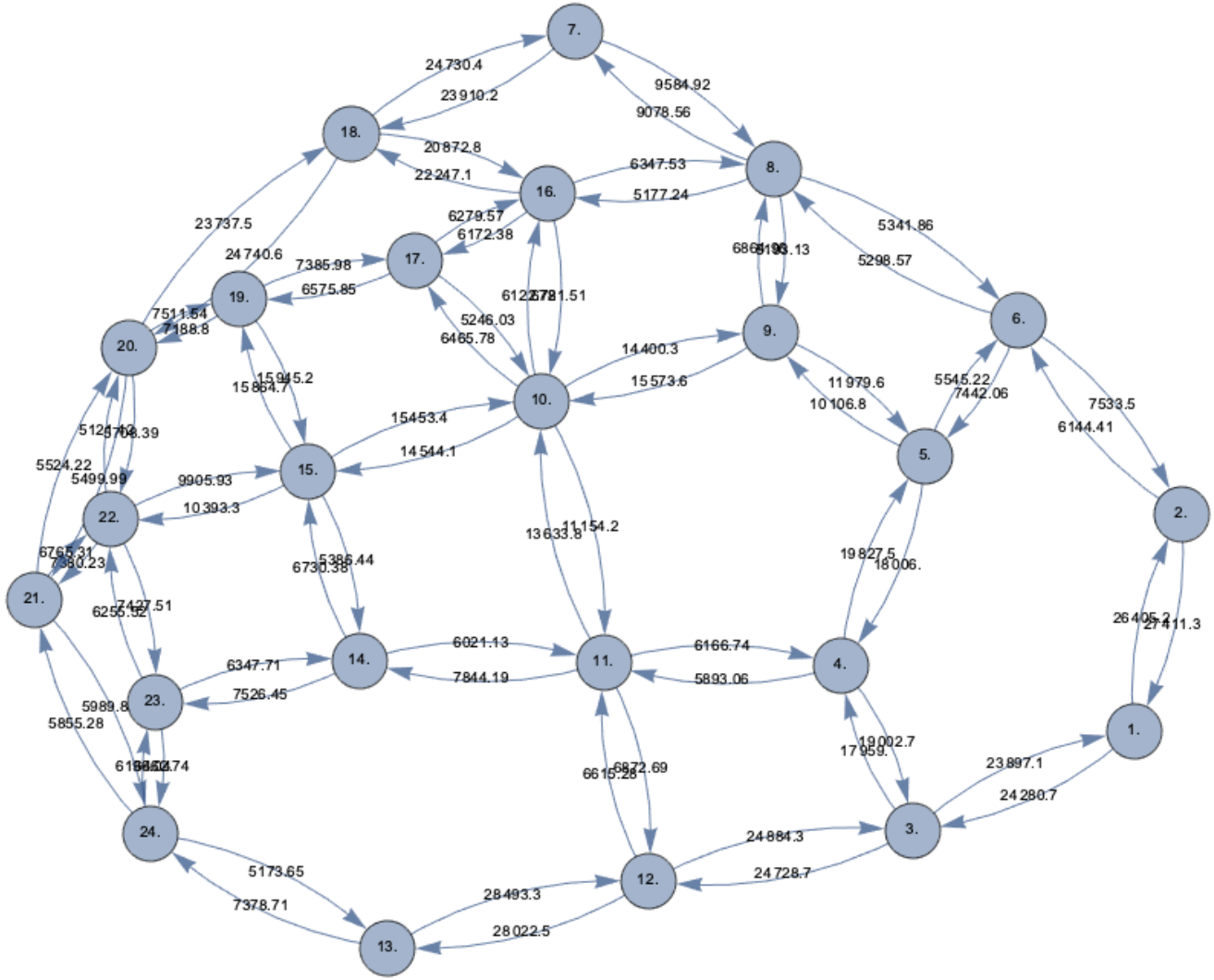
0,00	25900,20	23403,47	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
25900,20	0,00	0,00	0,00	0,00	4958,18	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
23403,47	0,00	0,00	17110,52	0,00	0,00	0,00	0,00	0,00	0,00	0,00	23403,47	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
0,00	0,00	17110,52	0,00	17782,79	0,00	0,00	0,00	0,00	0,00	4908,83	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
0,00	0,00	0,00	17782,79	0,00	4948,00	0,00	0,00	10000,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
0,00	4958,18	0,00	0,00	4948,00	0,00	0,00	4898,59	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	7841,81	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	23403,47	0,00	

0,00	0,00	0,00	0,00	0,00	4898,59	7841,81	0,00	5050,19	0,00	0,00	0,00	0,00	0,00	0,00	5045,82	0,00	0,00	0,00
0,00	0,00	0,00	0,00	10000,00	0,00	0,00	5050,19	0,00	13915,79	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	13915,79	0,00	10000,00	0,00	0,00	0,00	13512,00	4854,92	4993,51	0,00	0,00
0,00	0,00	0,00	4908,83	0,00	0,00	0,00	0,00	0,00	10000,00	0,00	4908,83	0,00	4876,51	0,00	0,00	0,00	0,00	0,00
0,00	0,00	23403,47	0,00	0,00	0,00	0,00	0,00	0,00	0,00	4908,83	0,00	25900,20	0,00	0,00	0,00	0,00	0,00	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	25900,20	0,00	0,00	0,00	0,00	0,00	0,00	5091,26
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	4876,51	0,00	0,00	0,00	5127,53	0,00	0,00	0,00	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	13512,00	0,00	0,00	0,00	5127,53	0,00	0,00	0,00	0,00	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	5045,82	0,00	4854,92	0,00	0,00	0,00	0,00	0,00	0,00	5229,91	19679,90	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	4993,51	0,00	0,00	0,00	0,00	0,00	5229,91	0,00	0,00	0,00
0,00	0,00	0,00	0,00	0,00	0,00	23403,47	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	19679,90	0,00	0,00	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	14564,75	0,00	4823,95	0,00	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	23403,47	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	4885,36
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	9599,18	0,00	0,00	0,00	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	4924,79	0,00	0,00	0,00	0,00	5078,51

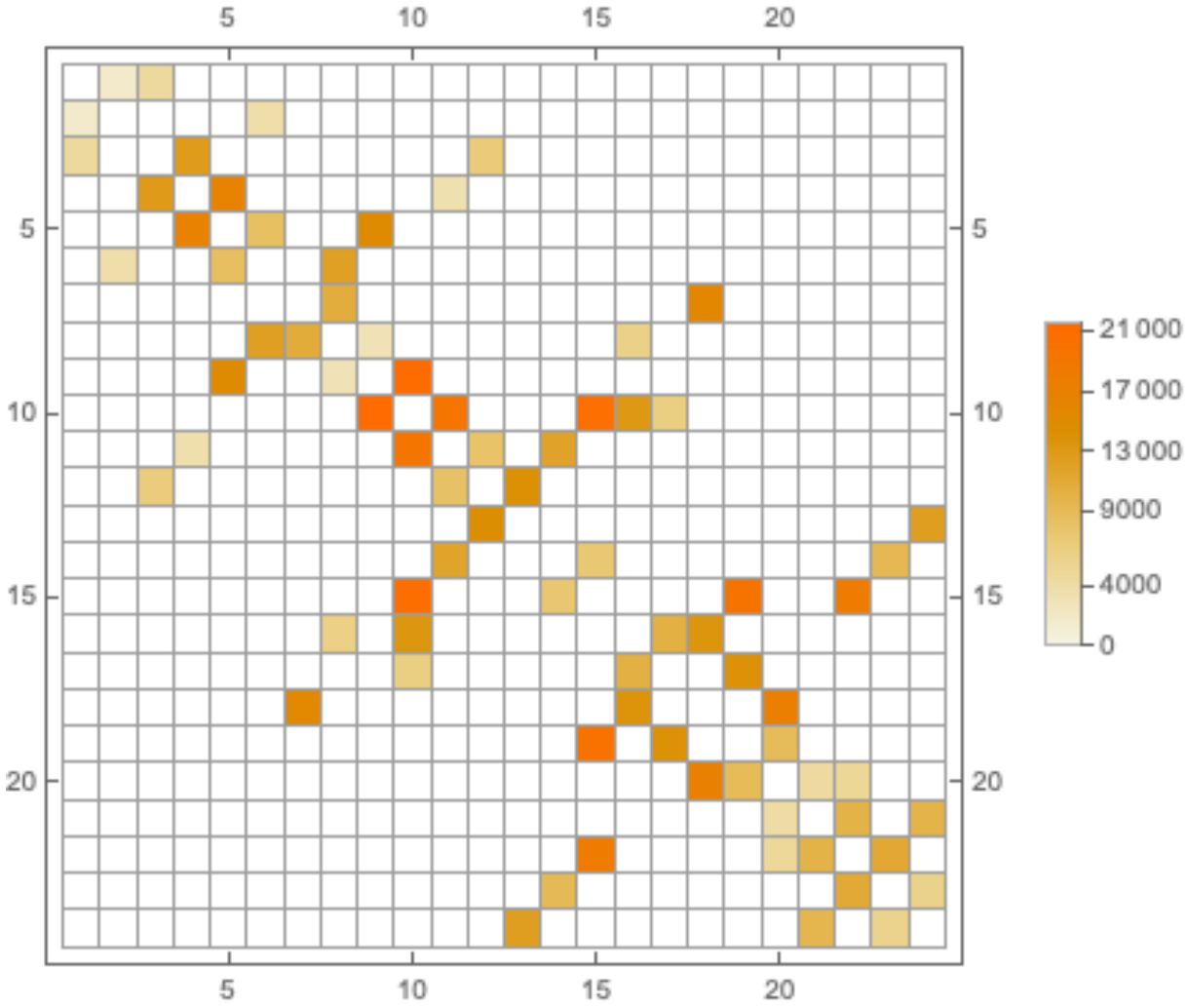
Матрица G - матрица ребер графа:

1	2	11	12
1	3	11	14
2	6	12	13
3	4	13	24
3	12	14	15
4	5	14	23
4	11	15	19
5	6	15	22
5	9	16	17
6	8	16	18
7	8	17	19
7	18	18	20
8	9	19	20
8	16	20	21
9	10	20	22
10	11	21	22
10	15	21	24
10	16	22	23
10	17	23	24

Приложение В. Полученное равновесное распределение максимальной пропускной способности дуг



Приложение Г. Равновесное распределение загрузки узлов сети



Приложение Д. Информация об изменении лучшего решения, найденного каждой частицей